Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Využití proximal bundle metody pro nehladkou optimalizaci

On using of the proximal bundle method for nonsmooth optimization

2015 Nikola Plívová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání diplomové práce

Student: **Bc. Nikola Plívová**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 1103T031 Výpočetní matematika

Téma: Využití proximal bundle metody pro nehladkou optimalizaci
On using of the proximal bundle method for nonsmooth optimization

Zásady pro vypracování:

Úloha minimalizace nediferencovatelné (nehladké) funkce se objevuje při řešení některých úloh z mechaniky, např. kontaktní úlohy se třením, či úloh tvarové optimalizace. Tento problém je možné vyřešit buď přímo pomocí algoritmu vhodného pro minimalizaci nediferencovatelných funkcí, nebo nepřímo spojitě diferencovatelnou aproximací funkce a následně pomocí algoritmu vhodného pro minimalizaci spojitě diferencovatelných funkcí. V této práci se věnujeme první možnosti. Pro minimalizaci nediferencovatelných funkcí bylo vyvinuto v uplynulých desetiletích mnoho algoritmů. Mezi nejvíce používané patří subgradientní metody a zejména bundle metody. Bundle metody mají řadu variant. Cílem práce je jedna z nich – proximal bundle metoda. Práce se zaměřuje na pochopení této metody a její následnou efektivní implementaci. V závěru práce se budeme věnovat testování algoritmu na vhodných úlohách.

Práce bude mít tyto části:
Úloha nehladké optimalizace
Clarkeův kalkul
Algoritmy pro minimalizaci nediferencovatelných funkcí
Proximal bundle metoda
Implementace proximal bundle metody
Testování algoritmu na vhodných úlohách

Seznam doporučené odborné literatury:

M.M.Mäkelä, P.Neittaanmäki: Nonsmooth optimization. Singapore, World Scientific, 1992.

J. Zowe: Nondifferentiable optimization: A motivation and a short introduction into the subgradient and the bundle concept. In: NATO SAI Series, 15, Computational Mathematical Programming, Schittkowski, K., (ed.), Springer-Verlag, New York, 1985, pp. 323-356.

F.H. Clarke: Optimization and Nonsmooth Analysis. New York, John Wiley, 1983.

Dále dle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Petr Beremlijski, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

"Pure mathematics is, in its way, the poetry of logical ideas."

– Albert Einstein

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Orlové dne 4.května 2015

.....

I would like to thank Ing. Petr Beremlijski, PhD. for his patience, very valuable advice and constant support, which help to create this diploma thesis.
Also many thanks go to my family and boyfriend for their understanding.

Abstract:
In practical applications we often have to solve the problem where the cost function is not differentiable. Thus we cannot directly use methods like Gradient Descent or Newton's methods. The aim of this thesis is to introduce method and implement an algorithm for nonsmooth optimization, namely the Proximal Bundle one. After briefly familiarization with a general optimization problem, we devote several chapters to nonsmooth analysis and optimization theory. The penultimate chapter consists of relatively detailed description of Proximal Bundle method algorithm and in the last part we summarize some numerical results obtained by our implementation in MATLAB.

Keywords:
Optimization, nonsmooth function, nonsmooth optimization, Subgradient method, Proximal Bundle method.

Abstrakt:
V praktických aplikacích se často setkáváme s problémy, kdy cenová funkce není diferencovatelná. V tomto případě nelze přímo využít metod, jako jsou například gradientní či Newtonova metoda. Cílem této práce je představit metodu a implementovat algoritmus pro nehladkou optimalizaci, konkrétně metodu Proximal Bundle. Po krátkém seznámení se s obecnou optimalizační úlohou, věnujeme několik kapitol teorii nehladké analýzy a optimalizace. Předposlední kapitola obsahuje relativně detailní popis algoritmu Proximal Bundle metody a v poslední kapitole jsou shrnuty výsledky numerických experimentů naší implementace v MATLABU.

Klíčová slova:
Optimalizace, nehladká funkce, nehladké optimalizace, Subgradientní metoda, Proximal Bundle metoda.

Contents

List of Tables

List of Figures

Listings

List of symbols

$\mathbb{R}^n$	n-dimensional Euclidean space
A	$n \times n$ matrix
A^{-1}	inverse matrix
A^T	transposed matrix
b, v	n-dimensional vectors
$C^1(\Omega)$	space of function with continuous first derivative
$C^2(\Omega)$	space of function with continuous second derivative
$\nabla f(x)$	gradient of function f
$f'(x)$	the first derivation of f
$f''(x)$	the second derivation of f
$H_{f(x)}$	Hessian matrix
$conv(\Omega)$	convex hull of set Ω
$\partial f(x)$	general gradient of function f
g_f	subgradient of function f
$\partial f(x)_\varepsilon^G$	the Goldstein ε-subdifferential of function f
$f'(x; v)$	directional derivative of f at x in the direction v
$f^\circ(x; v)$	general directional derivative of f at x in the direction v
$t \downarrow 0$	$t \to 0^+$
$\mathcal{P}(\mathbb{R}^n)$	set of all subsets in $\mathbb{R}^n$
$B(x; \varepsilon)$	open ball with radius ε and central point x
$N_\Omega(x)$	normal cone of the convex set Ω at x
$\arg\min f(x)$	a point where f has its minimum
$\|x\|$	Euclidean norm
J_f^k	index set
$\tilde{J}_f^k$	subset on an index set J_f^k
$\limsup_{x \to x_0} f(x)$	an upper limit of function f

1

Introduction

The classical optimization methods are used for differentiable functions. Unfortunately, we sometimes have situation where the cost function is not necessarily differentiable. This obstacle is solved variously: e.g. approximation by smooth function, which also brings us other difficulties in the form of approximation errors. That is one of the reasons why the theory of nonsmooth optimization was initiated.
The aim of this thesis is to introduce nonsmooth optimization method - namely Proximal Bundle method and implement an algorithm. The text is divided into five parts, which are put in this order. The first part consists of optimization theory principles. It is only an introduction to the optimization theory. We also describe here the differences between several types of the optimization programming. Some solution methods can be found there too.
Before we concentrate on the nonsmooth optimization, we have to devote to the nonsmooth analysis and optimization theory. The material of the second part consists of many definitions and theorems, which helps us to understand the requirements to the cost function property and optimality conditions. A failure of classical methods is described there, because this is another reason why we need to invent different solving methods.
In the third chapter, we consider a short introduction to the nonsmooth optimization and the solution methods - like Subgradient method and Bundle method. Since the Bundle method (Proximal Bundle method) is analysed in the next part, the Subgradient method occupies a larger part of this chapter. We can read there something about choosing the step size and about stopping rule. The influence of the step size choice is shown in the example.
The last two chapters are devoted to the Proximal Bundle method. After a survey of the optimization methods and analysis, we construct an algorithm for nonsmooth problems. We can find there a detailed algorithm description and a summary of our numerical results. We have tested some traditional examples.
Another important part of the thesis is a CD containing the algorithms of Subgradient and Proximal Bundle methods.

2

An optimization problem

The aim of this chapter is to introduce the basic principles of the optimization. Also we describe differences among the various types of the optimization and we write some information about the solution methods.
We draw inspiration from [1] and [2].

2.1 Introduction

An optimization problem in mathematics is a problem of finding the best element from all elements which are feasible. We use a cost function to decide which one is the best. In the general way, the cost function is a representation $f : \mathcal{D} \to \mathbb{R}$, where $\mathcal{D}$ is a set of elements from a suitable vector space. In many cases, we are trying to find extremes of some cost function in a feasible set $\Omega \subseteq \mathcal{D}$. As the minimum of function f is the same as the maximum of function $-f$, we can consider the general optimization problem as finding $\overline{x} \in \Omega$ such that

$$f(\overline{x}) \leq f(x), \quad x \in \Omega. \tag{2.1}$$

We can also write this problem in this way

$$\begin{cases} minimize & f(x) \\ subject\ to & x \in \Omega. \end{cases} \tag{2.2}$$

Both of this writing describes the same optimization problem. In our text we will use the second one. Solution of this problem is called a *global solution*.

For better imagination, we can mention as an example finding a steady state of small ball on a spring pendent over a obstacle. This problem can be represented as a minimization of potential energy function. Feasible set describes parts where small ball doesn't get through the obstacle. So we have to solve problem

$$\begin{cases} minimize & f(x) \\ subject\ to & x \in \Omega, \end{cases} \tag{2.3}$$

where $f(x) = x_1^2 + x_2^2 + m \cdot x_2$, $\Omega = \left\{x \in \mathbb{R} : \ x_1^2 + x_2 - 4 \geq 0, -x_1 - x_2 + 5 \geq 0\right\}$ and m is the ball weight.

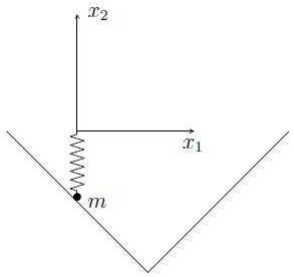

Figure 2.1: Small ball over the obstacle.

In the example below, we can see that the set Ω is restricted, so $\Omega \subset \mathcal{D}$ and $\Omega \neq \mathcal{D}$. This type of problem is called *constrained optimization*. In the case when $\Omega = \mathcal{D}$, we are talking about *unconstrained optimization*.

2.2 Classification of optimization methods

There are several types of optimization methods. It insists on a cost function f and also on function h, which describes equality or inequality. Both of these functions are defined at $\mathcal{D} \subset \mathbb{R}^n$, hence we are talking about smooth optimization.

2.2.1 Linear programming

General problem of linear programming is specified by linear function $b \in \mathbb{R}^n$ and by restriction $B \in \mathbb{R}^{m \times n}$ and $c \in \mathbb{R}^m$.

It can be written this way:

Find $\overline{x} \in \Omega$ such that $b^T\overline{x} \leq b^T x$ for all $x \in \Omega$,
where $\Omega = \{x \in \mathbb{R}^n, B_\varepsilon x = c_\varepsilon \text{ and } B_\chi x \leq c_\chi\}$

2.2.2 Quadratic programming

General problem of quadratic programming is different only in quadratic cost function, which is described in symmetrical matrix $A \in \mathbb{R}^{n \times n}$ and in $b \in \mathbb{R}^n$. This problem can be written like this:

> Find $\overline{x} \in \Omega$ such that $\frac{1}{2}\overline{x}^T A\overline{x} - b^T\overline{x} \leq \frac{1}{2}x^T Ax - b^T x$ for all $x \in \Omega$
> where Ω is described by linear equation mentioned in linear programming.

2.2.3 Nonlinear programming

When the cost function is neither linear nor quadratic but it is smooth with smooth first derivation at least, we are talking about nonlinear programming. If $\Omega \in \mathbb{R}^n$, so the area is not restricted, we have special type of nonlinear programming where we can use special algorithms. When we don't know anything about cost function and about binds, we have a very difficult problem which can be solved only by using algorithms, that have specified characteristic.

2.2.4 Nonsmooth programming

Nonsmooth programming is used when the cost function and the bind function are smooth except a small subset of domain. This partial smoothness can be employed for creating effective algorithms. This type of problem is commonly more difficult then nonlinear ones. First difficulties is in complicated description of minimal condition because the function needn't be differentiate in solution.

2.3 Solution methods

There are many methods which can be used for searching minima of function. Every method has its own request, so it helps us to decide which one we will use to solve some problem. Usually we want the function f to be continuous ($f \in C(\Omega)$) also to have continuous its first ($f \in C^1(\Omega)$) (the best the second ($f \in C^2(\Omega)$)) derivation. In the case when the function $f \in C^1(\Omega)$, we can use gradient of the function and in the case when $f \in C^2(\Omega)$, we use Hessian and Newton methods. Both of this methods will be shortly described below.

2.3.1 Gradient descent method

This method uses the negative direction of the gradient at the given point to find a local minimum of a function. It's a first-order optimization algorithm because it employs only first derivative of a function, so this function has to be $C^1(\Omega)$. A basic algorithm is mentioned below for better understanding.

Algorithm 2.1: **Algorithm of gradient descent.**

Step 0: Input $x_0, \varepsilon > 0$.

Step 1: Set $k = 1$ and $x_k = x_0$.

Step 2:

while $\|\nabla f(x)\| < \varepsilon$

$d_k = -\frac{\nabla f(x)}{\|\nabla f(x)\|}$

$t_k = \arg\min_{t>0} f(x_k + t \cdot d_k)$

$x_{k+1} = x_k + t_k d_k$

$k = k + 1$

end

Step 3: Output x_k.

2.3.2 Newton's method

For $x \in \mathbb{R}$

This method is employed for finding better approximation to the roots of function. Using Newton's method, we are trying to find when $f(x) = 0$. Geometrically, $(x_{k+1}, 0)$ is the intersection with the x axis of the tangent to the graph of function f at $(x_k, f(x_k))$. We can find the next value of x using this equation

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{2.4}$$

Similar equation is used to find minimum of function f

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}, \tag{2.5}$$

because the derivative of function in extreme is zero. As you can see, the function f has to be in $C^2(\Omega)$, so the condition are quite strict.
Also we can derive this method from the second order Taylor expansion $f_T(x)$ of a function f around x_n, where x_n is a sequence (constructed with Newton's method from initial point) that converges toward exact result $\overline{x}$. This $\overline{x}$ is a stationary point of f, so we know, that $f'(\overline{x}) = 0$. Let $\triangle x = x - x_n$. Then we can write Taylor expansion

$$f_T(x_n + \triangle x) = f_T(x_n) = f(x_n) + f'(x_n) \triangle x + \frac{1}{2} f''(x_n) \triangle x^2. \tag{2.6}$$

We attain its extreme when derivative with respect to $\triangle x = 0$, so $\triangle x$ solves this equation

$$f'(x_n) + f''(x_n) \triangle x = 0 \Longrightarrow -\frac{f'(x_n)}{f''(x_n)} = \triangle x. \tag{2.7}$$

Now we know that function f is well approximated by its second order Taylor expansion. When the initial point x_0 is well chosen, then the sequence (x_n)

$$\triangle x = x - x_n = -\frac{f'(x_n)}{f''(x_n)}, \tag{2.8}$$

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, \text{ for } n = 0, 1, \ldots \tag{2.9}$$

will converge towards a root of f'.

Algorithm 2.2: **Algorithm of Newton's method.**

Step 0: Input $x_0, \varepsilon > 0$.
Step 1: Set $k = 1$ and $x_k = x_0 - \frac{f'(x_0)}{f''(x_0)}$.
Step 2:
while $\|x_k - x_{k-1}\| < \varepsilon$
$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$
$k = k + 1$
end
Step 3: Output x_k.

For $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}$

We can generalized Newton's method to more dimension if we replace the first derivative with gradient of function and the second derivative with the Hessian matrix $H_{f(x)}$.
So

$$x_{n+1} = x_n - [H_{f(x)}]^{-1} \cdot \nabla f(x_n), \text{ for } n = 0, 1, \ldots \tag{2.10}$$

3

Nonsmooth analysis

In this chapter, we get acquainted with the theoretical basis of nonsmooth analysis and with the attributes, which every tested function should have. Concurrently, we show why we cannot use algorithms introduced in Chapter 2 for a nonsmooth optimization.
We draw inspiration from [1], [3], [4] and [6], so more information can be found there.

3.1 Convex set

Definition 3.1.1. We denote by $\langle x, y\rangle$ the close line segment joining x and y by

$$\langle x, y\rangle = \{z \in \mathbb{R}^n | z = \lambda x + (1-\lambda)y \text{ for } 0 \leq \lambda \leq 1\}. \tag{3.1}$$

A set $\Omega \subset \mathbb{R}^n$ is called *convex* if $\langle x, y\rangle \subset \Omega$ for all x and y belonging to Ω.

3.2 Convex hull

Definition 3.2.1. Let $\Omega \subset \mathbb{R}^n$. Then $conv(\Omega)$ denotes *convex hull* of set Ω. (The set of all convex combinations of points in Ω.)

$$conv(\Omega) = \left\{ \sum_{i=1}^{n} \lambda_i x_i | n \in \mathbb{N}, \lambda \in \mathbb{R}^n, x_1, \cdots, x_n \in \Omega, \lambda_i \geq 0, \forall i, \sum_{i=1}^{n} \lambda_i = 1 \right\} \tag{3.2}$$

3.3 Convex function

Definition 3.3.1. A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *convex* in $\mathbb{R}^n$ if

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y), \forall x, y \in \mathbb{R}^n, \forall \lambda \in \langle 0, 1\rangle. \tag{3.3}$$

3.4 Locally Lipschitz continuity

Definition 3.4.1. A function $f : \mathbb{R}^n \to \mathbb{R}$ is *locally Lipschitz continuous with constant L* at $x \in \mathbb{R}^n$ if there exists some constant $\varepsilon > 0$ such that

$$|f(y) - f(z)| \leq L\|y - z\|, \forall y, z \in B(x; \varepsilon). \tag{3.4}$$

Definition 3.4.2. A function $f : \Omega \subset \mathbb{R}^n \to \mathbb{R}$ is said to be *locally Lipschitz continuous* on Ω if there exists some constant $L = L(\Omega) > 0$ such that

$$|f(x) - f(y)| \leq L\|x - y\|, \forall x, y \in \Omega. \tag{3.5}$$

Definition 3.4.3. A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *locally Lipschitz continuous* in $\mathbb{R}^n$ if for every $\Omega \subset \mathbb{R}^n$, where Ω is a compact set, exists some constant $L = L(\Omega) > 0$ such that

$$|f(x) - f(y)| \leq L\|x - y\|, \forall x, y \in \Omega. \tag{3.6}$$

3.5 Generalized directional derivative

Definition 3.5.1. Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous at a point $x \in \mathbb{R}^n$. The *generalized directional derivative* of f at x in the direction of $v \in \mathbb{R}^n$ is defined by

$$f^\circ(x; v) = \limsup_{\substack{y \to x \\ t \downarrow 0}} \frac{f(y + tv) - f(y)}{t}. \tag{3.7}$$

3.6 General gradient and subgradient

Theorem 3.6.1. (Rademacher). Let $\Omega \subset \mathbb{R}^n$ be an open set and $f : \Omega \to \mathbb{R}$ be Lipschitz with constant K on Ω. Then f is differentiable almost everywhere on Ω.

Proof. Please see [4] or [6]. □

Definition 3.6.2. Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous (in $\mathbb{R}^n$). Then the *general gradient* of f at $x \in \mathbb{R}^n$ is the set

$$\partial f(x) = conv \left\{ g \in \mathbb{R}^n \middle| \begin{array}{c} g = \lim \nabla f(x_i), x_i \to x \\ \nabla f(x_i) \text{ exists}, \nabla f(x_i) \text{ converges} \end{array} \right\}. \tag{3.8}$$

Each element $g \in \partial f(x)$ is called a *subgradient* of f at x.

3.7 ε-subdifferential

ε-subdifferential is a modification of the classic subdifferential used in nonsmooth optimization. Now we give the definition and its basic properties.

Definition 3.7.1. Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous (in $\mathbb{R}^n$). Then *the Goldstein ε-subdifferential* is the set

$$\partial f(x)_\varepsilon^G = conv \left\{ g \in \mathbb{R}^n \middle| \begin{array}{c} g = \lim \nabla f(y_i), y_i \to y \\ \nabla f(y_i) \text{ exists}, \nabla f(y_i) \text{ converges}, y \in B(x;\varepsilon) \end{array} \right\}. \tag{3.9}$$

Each element $g \in \partial f(x)_\varepsilon^G$ is called *ε-subgradient* of the function f at x.

Theorem 3.7.2. Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous with constant L at x. Then

(i) $\partial f(x)_0^G = \partial f(x)$.

(ii) If $\varepsilon_1 < \varepsilon_2$, then $\partial f(x)_{\varepsilon_1}^G \subset \partial f(x)_{\varepsilon_2}^G$.

(iii) $\partial f(x)_\varepsilon^G$ is a nonempty, convex, compact set such that $\|g\| < L$ for all $g \in \partial f(x)_\varepsilon^G$.

(iv) The mapping $\partial f(\cdot)_\varepsilon^G : \mathbb{R}^n \to \mathcal{P}(\mathbb{R}^n)$ is upper semi-continuous.[1]

Proof. The proof can be found in [4]. □

Corollary 3.7.3. Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous at x. If $\varepsilon \geq 0$, then

$$\partial f(y) \subset \partial f(x)_\varepsilon^G \text{ for all } y \in B(x;\varepsilon). \tag{3.10}$$

Note. Written theory is for general function, but we can also find definitions and theorems especially for convex functions. Please see [4] or [6]. We don't need to mention it in this thesis, because we will solve problems with general cost functions.

[1] A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *upper semi-continuous* at x if for every sequence $\{x_i\}$ converging to x we have

$$\limsup_{i \to \infty} f(x_i) \leq f(x).$$

3.7.1 Example - general gradient and the Goldstein ε-subdifferential

Lets explain written theory on a simple example.
Find the general gradient and the Goldstein ε-subdifferential of function f at a given point x.

$$f(x) = |x-1| + |x| + |x+1|$$

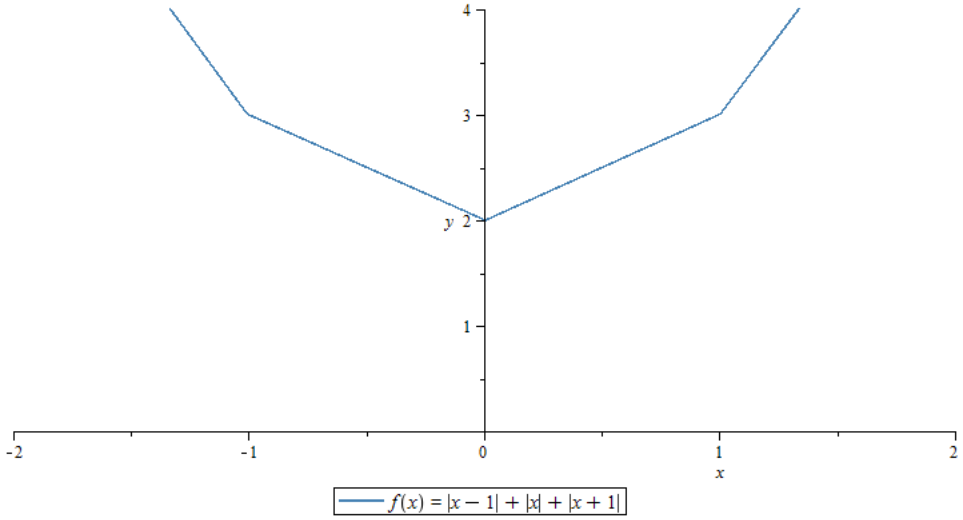

Figure 3.1: Graph of function f.

Another expression of the function f is

$$f(x) = \begin{cases} -3x & x \in (-\infty; -1) \\ -x+2 & x \in \langle -1; 0) \\ x+2 & x \in \langle 0; 1) \\ 3x & x \in \langle 1; \infty). \end{cases}$$

It helps us to find the general gradient and the Goldstein ε-subdifferential.

The general gradient

Write and plot the general gradient of f at $x = 0$. We will use definition 3.6.2. We have to use one-sided limit because $x = 0$ is a point where the function f is not smooth.
For

$$\left.\begin{array}{ll} x_i \to 0^+ : & g = 1 \\ x_i \to 0^- : & g = -1 \end{array}\right\} \partial f(x) = conv\{-1, 1\}$$

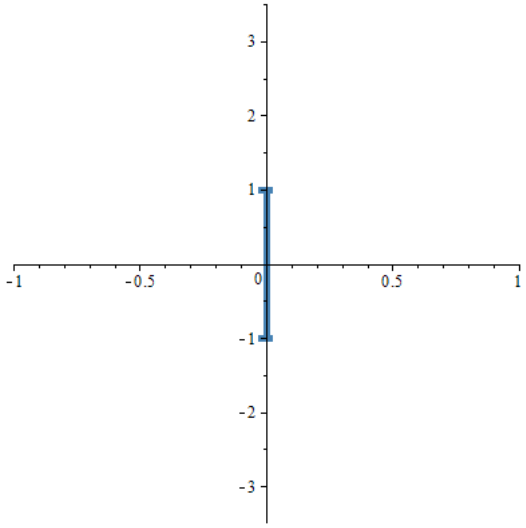

Figure 3.2: Graph of the general gradient of f at $x = 0$.

The general gradient of f for all $x \in \mathbb{R}$ is

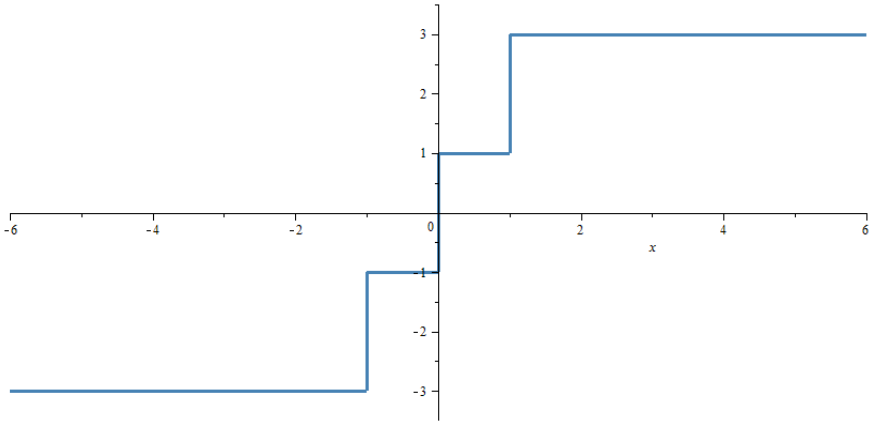

Figure 3.3: Graph of the general gradient of f.

The Goldstein ε-subdifferential

Write the Goldstein ε-subdifferential of f at $x = 0$ for different ε using theorem 3.7.1.
Set

- $\varepsilon_1 = 0.1$
  Then $y \in (-0.1, 0.1)$

$$\left.\begin{array}{lll} y \in (-0.1, 0): & y_i \to y \Rightarrow & g = -1 \\ y \in (0, 0.1): & y_i \to y \Rightarrow & g = 1 \\ y = 0: & y_i \to 0 \Rightarrow & g \in \langle -1, 1 \rangle \end{array}\right\} \partial f(x)^G_{\varepsilon_1} = conv\{-1, 1\}$$

  We can find the Goldstein ε-subdifferential of f at every $x \in (-0.1, 0.1)$. Here we can see a graph for all $x \in (-0.1, 0.1)$.

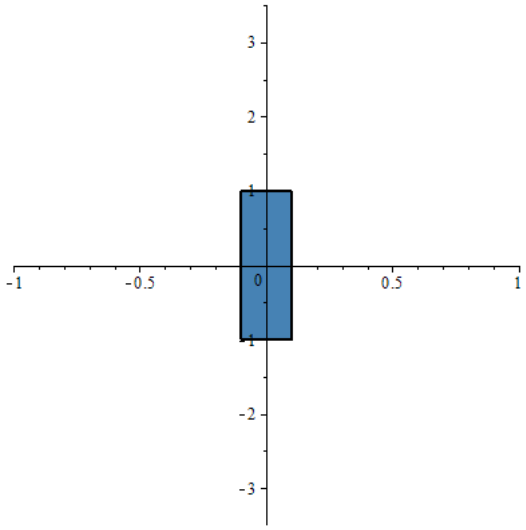

Figure 3.4: Graph of the Goldstein ε-subdifferential of f at $x \in \mathbb{R}$ for $\varepsilon = 0.1$.

- $\varepsilon_2 = 1.5$
  Then $y \in (-1.5, 1.5)$

$$\left.\begin{array}{lll} y \in (-1.5,-1): & y_i \to y \Rightarrow & g = -3 \\ y \in (-1,0): & y_i \to y \Rightarrow & g = -1 \\ y \in (0,1): & y_i \to y \Rightarrow & g = 1 \\ y \in (1,1.5): & y_i \to y \Rightarrow & g = 3 \\ y = -1: & y_i \to -1 \Rightarrow & g \in \langle -3,-1\rangle \\ y = 0: & y_i \to 0 \Rightarrow & g \in \langle -1,1\rangle \\ y = 1: & y_i \to 1 \Rightarrow & g \in \langle 1,3\rangle \end{array}\right\} \partial f(x)^G_{\varepsilon_2} = conv\{-3,-1,1,3\}$$

As in the example above, also here we can find the Goldstein ε-subdifferential of f at every $x \in (-1.5, 1.5)$. Here we can see a graph for all x.

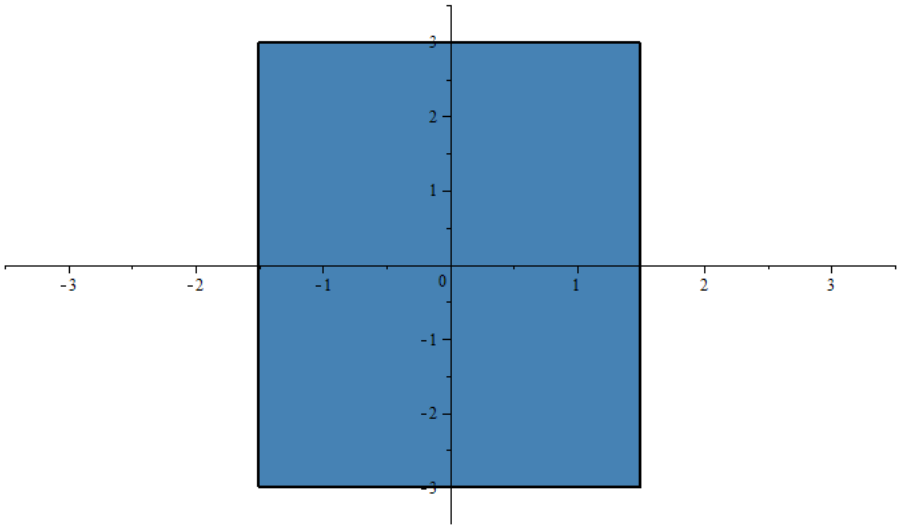

Figure 3.5: Graph of the Goldstein ε-subdifferential of f at $x \in (-1.5, 1.5)$ for $\varepsilon = 1.5$.

Here we can see an example of theorem 3.7.2 (ii)

$$\text{If } \varepsilon_1 < \varepsilon_2, \text{ then } \partial f(x)^G_{\varepsilon_1} \subset \partial f(x)^G_{\varepsilon_2}$$

Now we write the Goldstein ε-subdifferential at $x = 0.05$.

- $\varepsilon_1 = 0.1$
  Then $y \in (-0.05, 0.15)$

$$\left.\begin{array}{lll} y \in (-0.05, 0): & y_i \to y \Rightarrow & g = -1 \\ y \in (0, 0.15): & y_i \to y \Rightarrow & g = 1 \\ y = 0: & y_i \to 0 \Rightarrow & g \in \langle -1, 1 \rangle \end{array}\right\} \partial f(x)^G_{\varepsilon_1} = conv\{-1, 1\}$$

The Goldstein ε-subdifferential of f for all $x \in \mathbb{R}$ and $\varepsilon = 0.1$ is

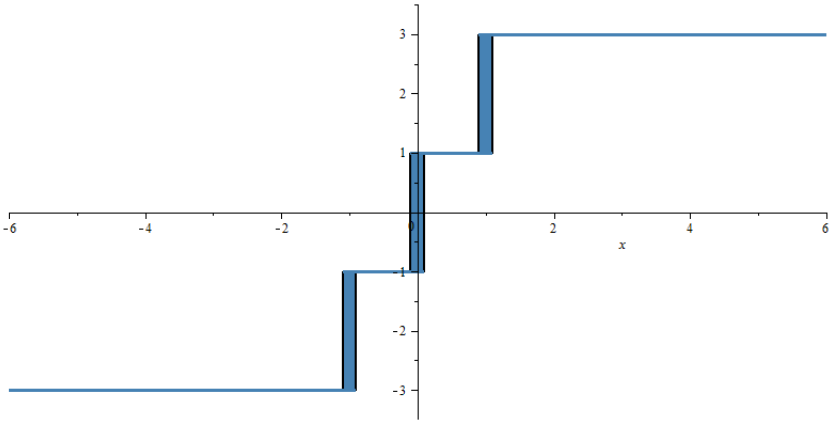

Figure 3.6: Graph of the Goldstein ε-subdifferential of f.

3.8 Optimality conditions

We should specify the basic necessary conditions at minimizer x for unconstrained problem. Also we have to note, that for convex functions these conditions are sufficient for x which is the minimum of function. All the theorems and proofs were inspired or taken from [4].

Theorem 3.8.1. If $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz at x and attains its local minimum at x, then

(i) $0 \in \partial f(x)$ and

(ii) $f^\circ(x; v) \geq 0$ for all $x \in \mathbb{R}^n$.

Proof. Suppose first that f attains a local minimum at x.Then there exists $\varepsilon > 0$ such that $f(x + tv) - f(x) \geq 0$ for all $0 < t < \varepsilon$ and $v \in \mathbb{R}^n$. Now we have

$$f^\circ(x; v) = \limsup_{\substack{y \to x \\ t \downarrow 0}} \frac{f(y + tv) - f(y)}{t} \geq \limsup_{t \downarrow 0} \frac{f(x + tv) - f(x)}{t} \geq 0$$

hands

$$f^\circ(x; v) \geq 0 = 0^T v \text{ for all } v \in \mathbb{R}^n,$$

which means by the definition of subdifferential that $0 \in \partial f(x)$. □

Theorem 3.8.2. If $f : \mathbb{R}^n \to \mathbb{R}$ is convex, then the following condition are equivalent:

(i) f attains its global minimum at x,

(ii) $0 \in \partial f(x)$,

(iii) $f'(x; v) \geq 0$ for all $x \in \mathbb{R}^n$.[1]

Proof. The assertion (ii) follows from (i) by Theorem 3.8.1(i). Supppose next, that (ii) holds. By Theorem 2.1.5. in [4] we have for all $v \in \mathbb{R}^n$

$$f'(x; v) = \max\{\xi^T v | \xi \in \partial f(x)\} \geq 0^T v = 0.$$

To complete the proof it suffices now to show that (iii) implies (i). To see this, consider $x' \in \mathbb{R}^n$. We have

$$f'(x; x' - x) = \max\{\xi^T (x' - x) | \xi \in \partial f(x)\}$$

and so there exists $\xi_{x'} \in \partial f(x)$ such that

$$0 \leq f'(x; x' - x) = \xi_{x'}^T (x' - x).$$

Then it follows that

$$f(x') \geq {}^2 f(x) + \xi_{x'}^T (x' - x) \geq f(x),$$

which means that f attains its global minimum at x. □

[1] If the function f is convex, then $f^\circ(x; v) = f'(x; v)$.

[2] We use the assumption that f is convex.

Next theorems are listed without proofs. To see them please look into [4], where all the omitted proofs can be found.

Theorem 3.8.3. If $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz at x and attains its local minimum at x, then

$$0 \in \partial_\varepsilon^G f(x). \tag{3.11}$$

Theorem 3.8.4. If $f : \mathbb{R}^n \to \mathbb{R}$ is convex, then the following condition are equivalent:

(i) $0 \in \partial_\varepsilon^G f(x)$,

(ii) x minimizes f within ε, i.e. $f(x) \leq f(y) + \varepsilon$ for all $y \in \mathbb{R}^n$.

We need this following theorem to write optimality conditions for constrained problems.

Theorem 3.8.5. *The normal cone* of the convex set Ω at $x \in \Omega$ is the set

$$N_\Omega(x) = \{z \in \mathbb{R}^n | (x' - x)^T z \leq 0 \text{ for all } x' \in \Omega\} \tag{3.12}$$

Now we can write conditions for constrained problems.

Theorem 3.8.6. If f is locally Lipschitz at x and attains its local minimum over the set $\Omega \subset \mathbb{R}^n$ at x, then

$$0 \in \partial f(x) + N_\Omega(x). \tag{3.13}$$

Theorem 3.8.7. If $f : \mathbb{R}^n \to \mathbb{R}$ is convex and the set Ω is convex, then the following condition are equivalent:

(i) $0 \in \partial f(x) + N_\Omega(x)$,

(ii) f attains its global minimum over Ω at x.

3.9 Failure of classical methods

In a classical smooth method, we usually use linear or quadratic model which leads to steepest descent or to Newton's method. Obviously, these two methods are not useful if a function is not smooth. There are two types of problem when classical methods fail.

3.9.1 Convergence to the nonoptimal point

Both of mentioned methods don't work at a kink of function. It can caused that we will be trapped at a nonoptimal kink and methods give us bad results. It is evident in this example

$$\min_{x\in\mathbb{R}} f(x) = \begin{cases} x^2, & x \geq 0 \\ x^2 + 2x, & x < 0 \end{cases} \tag{3.14}$$

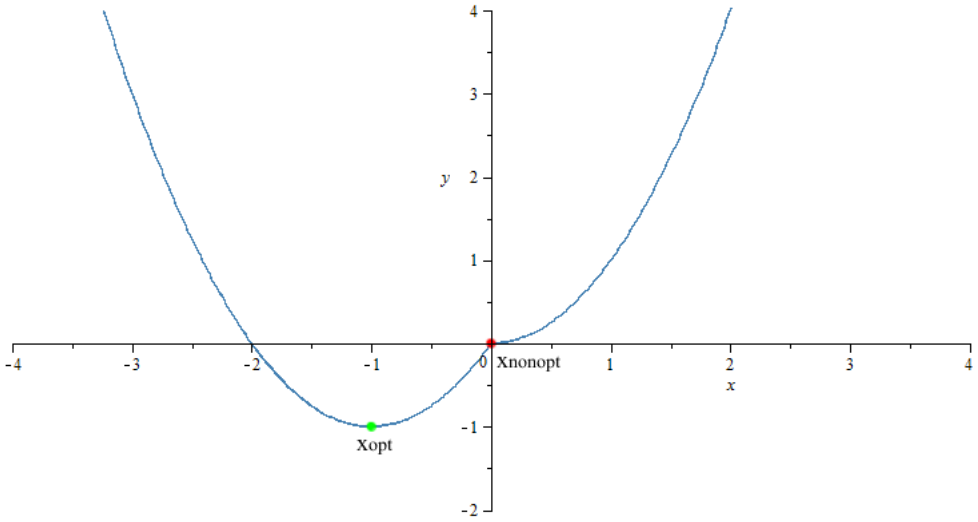

Figure 3.7: Convergence to the nonoptimal point.

Set the given point $x_0 = 2$. After few iterations, methods will give us result $x = 0$, but as we can see this is not the minimum of function. The optimal point is in $x = -1$.

3.9.2 Lacking of an implementable stopping rule

Let $f \in C^1(\Omega)$. The the gradient will be small in norm when we approach some optimal point.

$$|\nabla f(x_k)| \leq \varepsilon \quad (\varepsilon > 0 \text{ small})$$

This can help us to implement some stopping rule. But let see at this example

$$\min_{x \in \mathbb{R}} f(x) = |x| \tag{3.15}$$

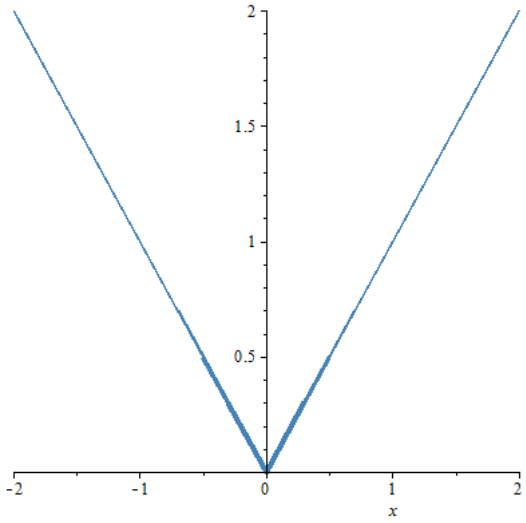

Figure 3.8: Lacking of an implementable stopping rule.

At each $x_k \neq 0$, when x_k is close to the optimal kink $\overline{x} = 0$, we have $|\nabla f(x_k)| = 1$. The steepest descent method will not stop close to the $\overline{x}$ because the gradient is not close to zero.

4

Nonsmooth optimization

In this part, we consider the nonsmooth optimization. After a very short description, we write there about solution methods like Subgradient method and Bundle methods and we give en example of using Subgradient method.
This chapter is inspired by [5] and [4].

4.1 Introduction

Let the cost function $f : \mathbb{R}^n \to \mathbb{R}$ to be only locally Lipschitz function on the feasible set $\Omega \subset \mathbb{R}^n$. Consider also the optimization problem

$$\begin{cases} minimize & f(x) \\ subject\ to & x \in \Omega, \end{cases} \tag{4.1}$$

which is nonlinear and constrained. When f is continuously differentiable then the problem (4.1) is called smooth and if $\Omega = \mathbb{R}^n$ then it is unconstrained.
Optimization methods are iterative. This means, that it begins at a given initial point $x_1 \in \mathbb{R}^n$ and then it constructs a sequence $\{x_i\}_{i=1}^{\infty} \subset \mathbb{R}^n$ which intends to converge to the solution we required. A general basic iterative algorithm is mentioned below. (Taken from [4])

Algorithm 4.1: **Basic Algorithm.**

Step 0: (Initialization) Find a feasible starting point $x_1 \in \Omega$ and set $k = 1$.

Step 1: (Direction finding) Find a feasible descent direction $d_k \in \mathbb{R}^n$:

$f(x_k + td_k) < f(x_k)$ and $x_k + td_k \in \Omega$ for some $t > 0$.

Step 2: (Stopping criterion) If x_k is "close enough" to the required solution then STOP.

Step 3: (Line search) Find a step size $t_k > 0$ such that

$t_k \approx \arg\min_{t>0}\{f(x_k + td_k)\}$ and $x_k + td_k \in \Omega$

Step 4: (Updating) Set $x_{k+1} = x_k + t_k d_k, k = k + 1$ and go on to Step 1.

4.2 Solution methods

As you can read in the part 2.3, we can find solution by using first order or second order methods when the cost function is differentiable. In this thesis, we are especially interested in non-differentiable functions, so we can't use mentioned methods and we have to find another way to solve this problem. Which methods can be employed, we describe in this part.

4.2.1 Subgradient method

Suppose for now the cost function f is smooth. In a standard first order or second order method, we make a feasible descent direction along the negative gradient $(-\nabla f(x))$ or along the negative gradient scaled by Hessian matrix $\left((-\nabla^2 f(x))^{-1} \cdot \nabla f(x)\right)$. For nonsmooth, locally Lipschitz cost function the gradient shouldn't exists in every point of the function, but we have at least one subgradient. Now it is obvious, how we can substitute the gradient. The feasible descent direction d_k in the basic algorithm 4.1 can be find in this way $d_k = \frac{g_k}{\|g_k\|}$ (g_k is subgradient at x_k) and then the updating part will be

$$x_{k+1} = x_k + t_k \frac{g_k}{\|g_k\|}, k = k + 1.$$

Unfortunately, this simple idea poses two critical problems: how we can choose the step size t_k and is there any implementable stopping rule?

Choosing the step size t_k

Let begin with the first task: How to choose the step size t_k? We have to consider two cases. In the first one, we know function value of some optimal point $\overline{x}$ and in the second one the function value in optimal point is not known.

Function value in optimal point is known

In this case, the step size t_k can be chosen this way:

- Also we can compute this special step size which guarantees a monotonous decrease for all k

  $t_k = \frac{\lambda(f(x_k)-f(\overline{x}))}{\|g_k\|}$ where $0 < \lambda < 2$.

We don't know $f(\overline{x})$ a priori.

- Constant step size: $t_k = const$
- A priori chosen step size which has to fulfil conditions:

  i) $t_k \to 0^+$;

  ii) $\sum_{k=0}^{\infty} t_k = \infty$.

- $t_k = M\, p^k$ where $M > 0$ and $0 < p < 1$

This is only the sum of the ways how to set t_k, to read more about this issue see [5].

Stopping rule

Unfortunately, the cost function is not smooth, so the gradient shouldn't become smaller as soon as we are closer to the optimal point. As we illustrated it in the part 3.9.2, it is very difficult to implement any stopping rule. This is one of the negative attribute.

4.2.2 Example - Subgradient method

Minimize $f(x), x \in \mathbb{R}^2$ where

$$f(x) = |x_1| + |x_2|. \tag{4.2}$$

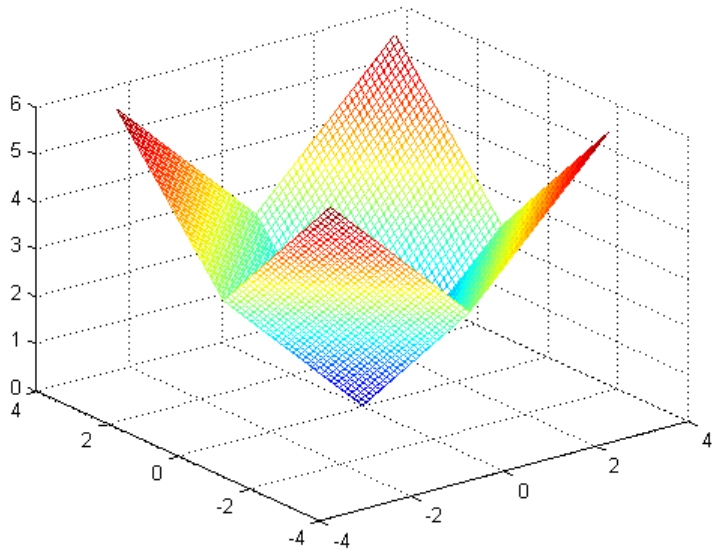

Figure 4.1: Graph of function $f = |x_1| + |x_2|$.

Solution

To find an optimal point, we use MATLAB m-files. First we need to implement some routine which helps us to compute subgradients at every point of a domain even at point of discontinuity. We have to split the domain into 4 parts:

I. $x_1, x_2 \geq 0$

II. $x_1 < 0$ and $x_2 \geq 0$

III. $x_1 \geq 0$ and $x_2 < 0$

IV. $x_1, x_2 < 0$

and in every part we need to count up partial derivatives

$$\text{I.} \quad \left.\begin{array}{l} \frac{\partial f}{\partial x_1} = 1 \\ \frac{\partial f}{\partial x_2} = 1 \end{array}\right\} = [1;1]$$

$$\text{II.} \quad \left.\begin{array}{l} \frac{\partial f}{\partial x_1} = -1 \\ \frac{\partial f}{\partial x_2} = 1 \end{array}\right\} = [-1;1]$$

$$\text{III.} \quad \left.\begin{array}{l} \frac{\partial f}{\partial x_1} = 1 \\ \frac{\partial f}{\partial x_2} = -1 \end{array}\right\} = [1;-1]$$

$$\text{IV.} \quad \left.\begin{array}{l} \frac{\partial f}{\partial x_1} = -1 \\ \frac{\partial f}{\partial x_2} = -1 \end{array}\right\} = [-1;-1]$$

Due to 3.6.2 we can assemble general gradient of $f(x)$

$$\partial f(x) = conv\left\{[1;1],[-1;1],[1;-1],[-1;-1]\right\}. \tag{4.3}$$

We implemented some routine which gives us value of subgradient corresponding to certain part. Now we can use mentioned m-files to find a minimum. We employed Subgradients method with step size $t_k = \frac{1}{k}$ (1st. type) and $t_k = \frac{\lambda(f(x_k)-f(\overline{x}))}{\|g_k\|}$, where $\lambda = 1$ and $\overline{x}$ is the optimal point (2nd. type).

Results can be seen in the following tables.

- *Set starting point* $x_0 = [-3;4]$ *and precision* $\varepsilon = 10^{-3}$

Type	Minimum	Number of iterations
(1st. type)	$[0; -0.7071 \cdot 10^{-3}]$	1001
(2nd. type)	$[-0.1110 \cdot 10^{-15}; -0.1110 \cdot 10^{-15}]$	4

Table 4.2: Results for $x_0 = [-3;4]$ and $\varepsilon = 10^{-3}$.

- *Set starting point* $x_0 = [-3; 4]$ *and precision* $\varepsilon = 10^{-5}$

Type	Minimum	Number of iterations
(1st. type)	$[0; -0.7071 \cdot 10^{-5}]$	100001
(2nd. type)	$[-0.1110 \cdot 10^{-15}; -0.1110 \cdot 10^{-15}]$	4

Table 4.3: Results for $x_0 = [-3; 4]$ and $\varepsilon = 10^{-5}$.

- *Set starting point* $x_0 = [7; -13]$ *and precision* $\varepsilon = 10^{-3}$

Type	Minimum	Number of iterations
(1st. type)	$[0.9992; -6.9992]$	1002
(2nd. type)	$[-0.4441 \cdot 10^{-15}; -0.4441 \cdot 10^{-15}]$	4

Table 4.4: Results for $x_0 = [7; -12]$ and $\varepsilon = 10^{-3}$.

- *Set starting point* $x_0 = [7; -13]$ *and precision* $\varepsilon = 10^{-5}$

Type	Minimum	Number of iterations
(1st. type)	$[0; -3.7439]$	100001
(2nd. type)	$[-0.4441 \cdot 10^{-15}; -0.4441 \cdot 10^{-15}]$	4

Table 4.5: Results for $x_0 = [7; -12]$ and $\varepsilon = 10^{-5}$.

Subgradient method (2nd. type) gave us results much faster then (1st. type), because this method uses better direction finding. In the next picture, we can see contours of $f(x)$, the starting point $[-3; 4]$ and the optimal point found by Subgradient method (2nd. type).

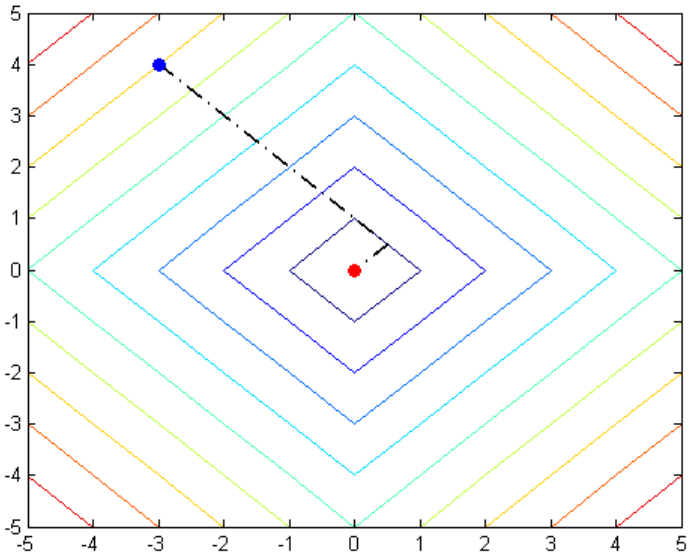

Figure 4.2: Graph of contours and solution.

4.2.3 Bundle methods

Another methods employed to solve problems with the nonsmooth cost function are Bundle methods, which were published in 1975 for the first time. These methods are working under the next precondition:

- In every $x \in \mathbb{R}^n$ we know $f(x)$ and at least one subgradient $g \in \partial f(x)$.

The main difference between Subgradient methods and Bundle methods are in these features:

- In every iteration the subgradient is gathered into a bundle (that's why these methods have their name)
  - If the chosen descent direction d_k doesn't give us a sufficient effect, we stay in the same iteration point $x_{k+1} = x_k$ but the subgradient is added into a bundle. This is called *null step*.
  - Otherwise we make a *serious step* - subgradient is also added into a bundle and we set $x_{k+1} = x_k + t_k d_k$.
- Through knowledge of the subgradient we can implement a stopping rule.

This is a brief introduction. In the next chapter we will devote specifically to Proximal Bundle method more in details.

5

Proximal Bundle method

As we mentioned above, this chapter is devoted to the Proximal Bundle method. A detailed algorithm description can be found there. This algorithm is divided into several parts - Direction finding, Line search, Weight update and the main Proximal Bundle algorithm.
It this chapter, we draw inspiration from [4], [6] and [7].

Consider the following problem

$$\left.\begin{array}{ll} minimize & f(x) \\ subject\ to & x \in \mathbb{R}^n \end{array}\right\} \qquad \textit{(P)}$$

where f is locally Lipschitz function. We suppose that we can evaluate at each $x \in \mathbb{R}^n$ subgradients $g^f \in \partial f(x)$ and the function value $f(x)$. Our task is to find the direction $d \in \mathbb{R}^n$, which solves

$$\left.\begin{array}{ll} minimize & f(x_k + d) - f(x_k) \\ subject\ to & d \in \mathbb{R}^n, \end{array}\right\} \qquad (5.1)$$

if the current iteration point $x_k \in \mathbb{R}^n$ is not optimal. We will employ the H function defined at $y \in \mathbb{R}^n$ by

$$H(x; y) = f(x) - f(y), \text{forall } x \in \mathbb{R}^n,$$

so the general problem *(P)* can be modified hereby

$$\left.\begin{array}{ll} minimize & H(x_k + d; x_k) \\ subject\ to & d \in \mathbb{R}^n. \end{array}\right\} \qquad \textit{(HP)}$$

5.1 Direction finding

For a while, we suppose that the problem *(P)* is convex. The nonconvex case will be considered later.

5.1.1 Derivation of the direction finding problem

Suppose that we have the current iteration point $x_k \in \mathbb{R}^n$, $x_k \neq x_1$(starting point), auxiliary point $y_j \in \mathbb{R}^n$ and subgradients $g_j^f \in \partial f(y_j)$ for $j \in J_f^k \subset \{1, \ldots, k\}$ where the index set J_f^k

is nonempty.
Let define the linearization at $x_k \in \mathbb{R}^n$ by

$$\overline{f_j}(x) = \overline{f}(x; y_j) = f(y_j) + (g_j^f)^T(x - y_j); \quad j \in J_f^k \tag{5.2}$$

Also we define the polyhedral approximation for all $x \in \mathbb{R}^n$ by

$$\hat{f}^k(x) = \max\{\overline{f}_j(x) | j \in J_f^k\} \tag{5.3}$$

where $\overline{f}_j(x)$ is represented in this way

$$\overline{f}_j(x) = f_j^k + (g_j^k)^T(x - x_k); \quad j \in J_f^k \tag{5.4}$$

denoted $f_j^k = \overline{f}_j(x_k)$. Considering the polyhedral approximation, we have to redefine function H as follows

$$\hat{H}^k(x) = \hat{f}^k(x) - f(x_k) \quad \text{for all } x \in \mathbb{R}^n. \tag{5.5}$$

Theorem 5.1.1. The polyhedral approximation function $\hat{H}^k(x)$ is convex. If in addition the problem (P) is convex, then

$$\hat{H}^k(x) \leq H(x; x_k) \quad \text{for all } x \in \mathbb{R}^n. \tag{5.6}$$

Proof. As maxima of affine functions the polyhedral approximations $\hat{f}^k(x)$ are convex, thus $\hat{H}^k$ is also convex. If the problem is convex, then (due to the definition of subgradient) linearizations $\overline{f}_j$ are lower approximations of the problem functions. □

Employing Proximal Bundle idea [1] with using the approximated function $\hat{H}^k$, we obtain the next approximation to *(HP)*

$$\left.\begin{array}{ll} minimize & \hat{H}(x_k + d) + \frac{u_k}{2}\|d\|^2 \\ subject\ to & d \in \mathbb{R}^n. \end{array}\right\} \qquad \textit{(GCP)}$$

We shall use following notation to balance the presentation

$$\alpha_{f,j}^k = f(x_k) - f_j^k \quad \text{for } j \in J_f^k \tag{5.7}$$

substituting f_j^k we obtain

$$\alpha_{f,j}^k = f(x_k) - f(y_j) - (g_j^f)^T(x_k - y_j) \quad \text{for } j \in J_f^k. \tag{5.8}$$

The *(GCP)* problem can be rewritten, but we have to employ the definition (5.3) which means, that

$$\hat{f}^k(x_k + d) \geq f(y_j) + (g_j^f)^T(x_k - y_j). \tag{5.9}$$

Furthermore, the function

$$\hat{H}^k(x_k + d) \geq \hat{f}^k(x_k + d) - f(x_k) \quad \text{for all } x_k + d \in \mathbb{R}^n. \tag{5.10}$$

[1] The idea of adding a penalty to limit the step length.

Then

$$\begin{aligned}\hat{H}^k(x_k+d) &\geq \hat{f}^k(x_k+d) - f(x_k) \geq \\ &\geq f(y_j) + (g_j^f)^T(x_k+d-y_j) - f(x_k) = \\ &= f(y_j) + (g_j^f)^T(x_k-y_j) + (g_j^f)^T d - f(x_k) = \\ &= -\alpha_{f,j}^k + (g_j^f)^T d.\end{aligned} \tag{5.11}$$

Denoting $v = \hat{H}^k(x_k+d)$, we rewrite the *(GCP)* problem this way

$$\left.\begin{array}{ll} minimize & v + \frac{u_k}{2}\|d\|^2 \\ subject\ to & -\alpha_{f,j}^k + (g_j^f)^T(d) \leq v \quad \text{for all } j \in J_f^k. \end{array}\right\} \tag{BP}$$

The *(BP)* problem in this form is called a primal (minimization) problem. In optimization theory we can also used a dual form of the problem. This duality means that the optimization problems can be viewed from a different perspective, where we supposed to find multipliers solving the dual problem. Let's try to derive the dual form of *(BP)*.

First, we should assemble Lagrange function of *(BP)*.

$$L(v,d,\lambda) = v + \frac{u_k}{2}\|d\|^2 + \sum_{j=1}^{k}\lambda_j(-\alpha_{f,j}^k + (g_j^f)^T d - v) \tag{5.12}$$

and also the Karush–Kuhn–Tucker conditions of *(BP)*.

1. $\nabla_{v,d} L(v,d,\lambda) = 0$
2. $\lambda_j(-\alpha_{f,j}^k + (g_j^f)^T d - v) = 0$ for all $j \in J_f^k$
3. $\lambda_j \geq 0$
4. $v \geq -\alpha_{f,j}^k + (g_j^f)^T d$

Applying the first condition we obtain

$$\nabla_{v,d} L(v,d,\lambda) = \begin{bmatrix} 1 - \sum\limits_{j=1}^{k}\lambda_j \\ u_k d + \sum\limits_{j=1}^{k}\lambda_j g_j^f \end{bmatrix} = 0 \tag{5.13}$$

then follows

$$1 - \sum_{j=1}^{k}\lambda_j = 0 \Rightarrow \sum_{j=1}^{k}\lambda_j = 1 \tag{5.14}$$

$$u_k d + \sum_{j=1}^{k}\lambda_j g_j^f = 0 \Rightarrow d = \frac{-\sum\limits_{j=1}^{k}\lambda_j g_j^f}{u_k}. \tag{5.15}$$

From the second condition we have

$$\begin{aligned}\sum_{j=1}^{k}\lambda_j(-\alpha_{f,j}^k+(g_j^f)^Td-v)=0\\ -\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k+\sum_{j=1}^{k}\lambda_j(g_j^f)^Td=\sum_{j=1}^{k}\lambda_j v.\end{aligned}\tag{5.16}$$

Substitute d from (5.15) into (5.16) we obtain

$$-\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k+\sum_{j=1}^{k}\lambda_j(g_j^f)^T\begin{pmatrix}\dfrac{-\sum\limits_{j=1}^{k}\lambda_j g_j^f}{u_k}\end{pmatrix}=v\sum_{j=1}^{k}\lambda_j\tag{5.17}$$

Using (5.14) we can express v

$$-\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k-\frac{1}{u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f\right\|^2=v.\tag{5.18}$$

Now we have expression for d and v, so we can substitute them into Lagrangian and we obtain

$$\begin{aligned}&\min_x\max_\lambda L(v,d,\lambda)\\ &\max_\lambda-\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k-\frac{1}{u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f\right\|^2+\frac{u_k}{2}\left\|\frac{-\sum\limits_{j=1}^{k}\lambda_j g_j^f}{u_k}\right\|^2\\ &\max_\lambda-\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k-\frac{2}{2u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f\right\|^2+\frac{1}{2u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f\right\|^2\\ &\max_\lambda-\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k-\frac{1}{2u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f\right\|^2\\ &\min_\lambda\frac{1}{2u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f\right\|^2+\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k\end{aligned}\tag{5.19}$$

So via dualization we can solve the next problem

$$\left.\begin{array}{ll}minimize & \frac{1}{2u_k}\left\|\sum\limits_{j=1}^{k}\lambda_j g_j^f\right\|^2+\sum\limits_{j=1}^{k}\lambda_j\alpha_{f,j}^k\\ subject\ to & \sum\limits_{j=1}^{k}\lambda_j=1\\ & \lambda_j\geq 0 \text{ for all } j\in J_f^k\end{array}\right\}\tag{DP}$$

5.1.2 Subgradient aggregation

There is still one task to solve. How can we choose the index set J_f^k? This set is a nonempty subset of $\{1, \ldots, k\}$. We have to solve many problem with storage and computation in practice. To avoid big memory requirement, we will employ the subgradient aggregation strategy. So we would like to aggregate the constraints generated by past subgradient, which helps us to keep the number of constraints within certain limits.

Definition 5.1.2. Let λ_j^k for $j \in J_f^k$ be Lagrange multipliers of the problem *(BP)* at iteration k. We define the aggregate subgradients by

$$p_f^k = \sum_{j \in J_f^k} \lambda_j^k g_j^f \tag{5.20}$$

$$\tilde{f}_p^k = \sum_{j \in J_f^k} \lambda_j^k f_j^k \tag{5.21}$$

$$\tilde{\alpha}_{f,p}^k = \sum_{j \in J_f^k} \lambda_j^k \alpha_{f,j}^k \tag{5.22}$$

and the aggregate linearization by

$$\tilde{f}_p(x) = \tilde{f}_p^k + (p_f^k)^T (x - x_k) \tag{5.23}$$

Lemma 5.1.3. The *(BP)* problem is equivalent to the reduced problem

$$\left.\begin{array}{ll} minimize & v + \frac{u_k}{2}\|d\|^2 \\ subject\ to & -\alpha_{f,j}^k + (g_j^f)^T(d) \le v \quad \text{for all } j \in \tilde{J}_f^k \\ & -\tilde{\alpha}_{f,p}^k + (p_f^k)^T(d) \le v \end{array}\right\} \tag{RP}$$

where $\tilde{J}_f^k$ is any subset of J_f^k.

Proof. Please see [4] □

Now we can solve the *(RP)* problem which seems to be appropriate for keeping the index set finite. However, it presents a problem as well. At the beginning of every iteration, the vector p_f^k is not known. This problem can be eliminated recursively. Let $x_1 \in \mathbb{R}^n$ be a feasible starting point, then the algorithm seems like

$$y_1 = x_1 \qquad p_f^0 = g_1^f \in \partial f(y_1) \qquad f_p^1 = f_1^1 = f(y_1) \qquad J = \{1\}.$$

At iteration k the unknown vectors $\tilde{f}_p^k, \tilde{p}_f^k$ will be replaced by the previously generated f_p^k, p_f^{k-1} and also we define

$$\alpha_{f,p}^k = f(x_k) - f_p^k \tag{5.24}$$

which leads to the problem

$$\left.\begin{array}{ll} minimize & v + \frac{u_k}{2}\|d\|^2 \\ subject\ to & -\alpha_{f,j}^k + (g_j^f)^T(d) \le v \quad \text{for all } j \in \tilde{J}_f^k \\ & -\alpha_{f,p}^k + (p_f^{k-1})^T(d) \le v \end{array}\right\} \tag{ABP}$$

Also here we can try to derive the dual form of *(ABP)*.

As before, we should assemble Lagrange function of *(ABP)* first

$$L(v,d,\lambda) = v + \frac{u_k}{2}\|d\|^2 + \sum_{j=1}^{k} \lambda_j(-\alpha_{f,j}^k + (g_j^f)^T d - v) + \lambda_p(-\alpha_{f,p}^k + (p_f^{k-1})^T(d) - v) \quad (5.25)$$

and also the Karush–Kuhn–Tucker conditions of *(ABP)*

1. $\nabla_{v,d} L(v,d,\lambda) = 0$
2. $\lambda_j(-\alpha_{f,j}^k + (g_j^f)^T d - v) = 0$ for all $j \in J_f^k$
3. $\lambda_p(-\alpha_{f,p}^k + (p_f^{k-1})^T(d) - v) = 0$
4. $\lambda_j, \lambda_p \geq 0$
5. $v \geq -\alpha_{f,j}^k + (g_j^f)^T d$
6. $v \geq -\alpha_{f,p}^k + (p_f^{k-1})^T(d)$

Applying the first condition we obtain

$$\nabla_{v,d} L(v,d,\lambda) = \begin{bmatrix} 1 - \sum\limits_{j=1}^{k} \lambda_j - \lambda_p \\ u_k d + \sum\limits_{j=1}^{k} \lambda_j g_j^f + \lambda_p p_f^{k-1} \end{bmatrix} = 0 \quad (5.26)$$

then follows

$$1 - \sum_{j=1}^{k} \lambda_j - \lambda_p = 0 \Rightarrow \sum_{j=1}^{k} \lambda_j + \lambda_p = 1 \quad (5.27)$$

$$u_k d + \sum_{j=1}^{k} \lambda_j g_j^f + \lambda_p p_f^{k-1} = 0 \Rightarrow d = \frac{-\sum\limits_{j=1}^{k} \lambda_j g_j^f - \lambda_p p_f^{k-1}}{u_k}. \quad (5.28)$$

Adding up the second and the third conditions we have

$$\begin{aligned} &\sum_{j=1}^{k} \lambda_j(-\alpha_{f,j}^k + (g_j^f)^T d - v) + \lambda_p(-\alpha_{f,p}^k + (p_f^{k-1})^T(d) - v) = 0 \\ &\quad - \sum_{j=1}^{k} \lambda_j \alpha_{f,j}^k + \sum_{j=1}^{k} \lambda_j (g_j^f)^T d - \lambda_p \alpha_{f,p}^k + \lambda_p (p_f^{k-1})^T d = \sum_{j=1}^{k} \lambda_j v + \lambda_p v \\ &- \sum_{j=1}^{k} \lambda_j \alpha_{f,j}^k - \lambda_p \alpha_{f,p}^k + d\left(\sum_{j=1}^{k} \lambda_j (g_j^f)^T + \lambda_p (p_f^{k-1})^T\right) = v\left(\sum_{j=1}^{k} \lambda_j + \lambda_p\right). \end{aligned} \quad (5.29)$$

Substituting d from (5.28) into (5.29) and using (5.27) we can express v

$$\begin{aligned} -\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k - \lambda_p\alpha_{f,p}^k - \frac{1}{u_k}\left(\sum_{j=1}^{k}\lambda_j g_j^f + \lambda_p p_f^{k-1}\right)\left(\sum_{j=1}^{k}\lambda_j g_j^f + \lambda_p p_f^{k-1}\right) &= v \\ -\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k - \lambda_p\alpha_{f,p}^k - \frac{1}{u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f + \lambda_p p_f^{k-1}\right\|^2 &= v \end{aligned} \tag{5.30}$$

Now we have expression for d and v, so we can substitute them into Lagrangian and we obtain

$$\begin{aligned} &\min_x \max_\lambda L(v,d,\lambda) \\ &\max_\lambda -\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k - \lambda_p\alpha_{f,p}^k - \frac{1}{u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f + \lambda_p p_f^{k-1}\right\|^2 + \frac{u_k}{2}\left\|\frac{-\sum_{j=1}^{k}\lambda_j g_j^f - \lambda_p p_f^{k-1}}{u_k}\right\|^2 \\ &\max_\lambda -\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k - \lambda_p\alpha_{f,p}^k - \frac{2}{2u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f + \lambda_p p_f^{k-1}\right\|^2 + \frac{1}{u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f + \lambda_p p_f^{k-1}\right\|^2 \\ &\max_\lambda -\sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k - \lambda_p\alpha_{f,p}^k - \frac{1}{2u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f + \lambda_p p_f^{k-1}\right\|^2 \\ &\min_\lambda \frac{1}{2u_k}\left\|\sum_{j=1}^{k}\lambda_j g_j^f + \lambda_p p_f^{k-1}\right\|^2 + \sum_{j=1}^{k}\lambda_j\alpha_{f,j}^k + \lambda_p\alpha_{f,p}^k \end{aligned} \tag{5.31}$$

So via dualization we can solve this problem

$$\left.\begin{array}{ll} minimize & \frac{1}{2u_k}\left\|\sum\limits_{j=1}^{k}\lambda_j g_j^f + \lambda_p p_f^{k-1}\right\|^2 + \sum\limits_{j=1}^{k}\lambda_j\alpha_{f,j}^k + \lambda_p\alpha_{f,p}^k \\ subject\ to & \sum\limits_{j=1}^{k}\lambda_j + \lambda_p = 1 \\ & \lambda_j, \lambda_p \geq 0. \end{array}\right\} \quad \textit{(ADP)}$$

5.1.3 Nonconvexity

For convex function the linearisation error increases as far as we are from a given point. We can't take advantage of this attribute when we have nonconvex function. Due to this fact, we have to generalize the linearisation errors. We shall employ so-called *subgradient locality measures*[1], which were described for the first time by Kiwiel.

[1] Subgradient locality measures and linearisation errors are identical for convex function.

Definition 5.1.4. We define *distance measure* at each iteration k by

$$s_j^k \begin{cases} \|x_j - y_j\| + \sum_{i=j}^{k-1} \|x_{i+1} - x_i\| & \text{for } j = 1, \dots, k-1, \\ \|x_k - y_k\| & \text{for } j = k, \end{cases} \tag{5.32}$$

the *aggregate distance measures* by

$$\begin{cases} \tilde{s}_k^f = \sum_{j\in} \tilde{\lambda}_j^k s_j^k + \tilde{\lambda}_p^k s_f^k, \\ s_{k+1}^f = \tilde{s}_k^f + \|x_{k+1} - x_k\|, \quad (s_1^f = 0) \end{cases} \tag{5.33}$$

and the *subgradient locality measures* by

$$\begin{cases} \beta_{f,j}^f = \max\{|\alpha_{f,j}^k|, \gamma_f (s_j^k)^2\} \quad \text{for all } j \in \tilde{J}_f^k, \\ \beta_{f,p}^f = \max\{|\alpha_{f,p}^k|, \gamma_f (s_f^k)^2\}, \\ \tilde{\beta}_{f,p}^f = \max\{|\tilde{\alpha}_{f,p}^k|, \gamma_f (\tilde{s}_f^k)^2\}, \end{cases} \tag{5.34}$$

where $\gamma_f \geq 0$ is the *distance measure parameter* ($\gamma_f = 0$) for convex function f.

5.2 Line search

Suppose that we have found a solution (d_k, v_k) of the *(ABP)* problem at the k-th iteration with the subgradient locality measures. Also, we have to remember that this is solution for $\hat{H}^k$ which only approximates H, so the chosen direction $x_{k+1} = x_k + d_k$ needn't to be the best one. therefore we will try to find a feasible step size $t_k \in (0, 1]$ such that

$$t_k \approx \arg \min_{t_k \in (0,1]} \{f(x_k + t d_k)\} \text{ and } x_k + t d_k \in \mathbb{R}^n. \tag{5.35}$$

Unfortunately, the direction d_k may not be necessarily the descent one or the descending is not sufficient. That's why we have to modify the general line search.
To detect discontinuities in the gradient of f, we will describe a two-point line search. We assume that

$$\begin{cases} m_L = (0, \frac{1}{2}) \\ m_R = (m_L, 1) \\ \bar{t} = (0, 1] \\ \zeta = 1 - \frac{1}{2} \cdot \frac{1}{1 - m_L} \end{cases} \tag{5.36}$$

are fixed parameters for line search.

Now we will search the largest number $t_L^k \in [0,1]$ such that

$$\begin{array}{l} \text{a) } f(x_k + t_L^k d_k) \leq f(x_k) + m_L t_L^k v_k \\ \text{b) } t_L^k \geq \bar{t} \end{array} \tag{5.37}$$

If such t_L^k exists we have

- a long serious step: $\begin{array}{l} x_{k+1} = x_k + t_L^k d_k \\ y_{k+1} = x_{k+1}, \end{array}$

if (5.37)(a) is held while $0 < t_L^k < \bar{t}$ then we have

- a short serious step: $\begin{array}{l} x_{k+1} = x_k + t_L^k d_k \\ y_{k+1} = x_k + t_R^k d_k \end{array}$

and if $t_L^k = 0$ we have

- a null serious step: $\begin{array}{l} x_{k+1} = x_k \\ y_{k+1} = x_k + t_R^k d_k \end{array}$

where $t_R^k > t_L^k$ is such that

$$\text{c) } -\beta_{f,k+1}^{k+1} + (g_{k+1}^f)^T d_k \geq m_R v_k \tag{5.38}$$

In long serious step, there is no reason to detect discontinuities in the gradient of f and we set $g_{k+1}^f \in \partial f(x_{k+1})$.
In short and null step, there is a discontinuity so the new subgradient will be $g_{k+1}^f \in \partial f(y_{k+1})$ and it will constrain a modification of the next direction finding problem.
At the next line of this thesis, we will show an algorithm of a line search method where conditions a) - c) are held.

Algorithm 5.1: **Algorithm Line search.**

Step 1: Set $t_L^k = 0$ and $t = t_U = 1$.

Step 2: If

$$f(x_k + td_k) \leq f(x_k) + m_L t v_k$$

set $t_L^k = t$, otherwise set $t_U = t$.

Step 3: If $t_L^k \geq \bar{t}$ set $t_R^k = t_L^k$ and STOP, otherwise calculate $g^f \in \partial f(x_k + td_k)$ and

$$\beta = \max\{|f(x_k + t_L^k d_k) - f(x_k + td_k) + (t - t_L^k)(g^f)^T d_k|, \gamma_f (t - t_L^k)^2 \|d_k\|^2\}.$$

Then if

$$-\beta + (g^f)^T d_k \geq m_R v_k,$$

set $t_R^k = t$ and STOP.

Step 4: If $t_L^k = 0$, then set

$$t = \max\{\zeta \cdot t_U, \frac{\frac{1}{2} t_U^2 v_k}{t_U v_k + f(x_k) - f(x_k + t_U d_k)}\}$$

and if $t_L^k > 0$, then set $t = \frac{1}{2} \cdot (t_L^k + t_U)$.

Step 5: Go on to *Step 2*.

5.3 Weight update

We cannot choose the weight u_k such that it is constant, because it brings several difficulties.

- If u_k is very large, we can obtain $|v_k|$ and $\|d_k\|$ too small. Due to this, all steps will be serious and decline will be small.
- If u_k is very small, then $|v_k|$ and $\|d_k\|$ is too large and every serious step is followed by many null steps.

So we need to keep it up as variable and change its value whet it is necessary.

In 1990 K. C. Kiwiel presented a safeguarded quadratic interpolation technique for updating u_k.
We will detect whether u_k is not too large after we have a long serious step ($t_L = 1$). As we can see in the section 5.2, we have $y_{k+1} = x_k + d_k$ and

$$f(y_{k+1}) \leq f(x_k) + m_L v_k \tag{5.39}$$

and if

$$f(y_{k+1}) \leq f(x_k) + m_R v_k \tag{5.40}$$

we will decrease u_k. To decrease it we will use this quadratic interpolation

$$u_k^{int} = 2u_k \left(1 - \left[f(y_{k+1}) - f(x_k)\right]/v_k\right). \tag{5.41}$$

In the case where condition (5.40) is not held, we also decrease u_k by updating

$$u_{k+1} = \max\left\{u_k/2, u_{min}\right\}. \tag{5.42}$$

We will test if

$$\beta_{f,k+1}^{k+1} > \max\left\{\|p_k\| + \tilde{\beta}_p^k, -10v_k\right\} \tag{5.43}$$

is held after we have many successive null steps. When this condition is fulfilled, the u_k is increased and updated by

$$u_{k+1} = \min\left\{u_{k+1}^{int}, 10u_k\right\} \tag{5.44}$$

Now we can show an algorithm presented by Kiwiel in [7].

Algorithm 5.2: **Algorithm Weight Update.**

Step 1: Set $u = u_k$.

Step 2: If $t_L \in (0, 1)$ STOP.

Step 3: Else if $t_L = 0$.

Set

$$\varepsilon_v^{k+1} = \min\left\{\varepsilon_v^k, \|p_k\| + \tilde{\beta}_p^k\right\}.$$

If $i_u^k < -3$ and

$$\beta_{f,k+1}^{k+1} > \max\left\{\varepsilon_v^{k+1}, -10v_k\right\},$$

set $u = u_{k+1}^{int}$.

Set

$$u_{k+1} = \min\left\{u, 10u_k\right\},$$
$$i_u^{k+1} = \min\left\{i_u^k - 1, -1\right\}.$$

If $u_{k+1} \neq u_k$ set $i_u^{k+1} = -1$. STOP.

Step 4: Else.

Set

$$\varepsilon_v^{k+1} = \max\left\{\varepsilon_v^k, -2v_k\right\}.$$

If $i_u^k > 0$ and

$$f(y_{k+1}) \leq f(x_k) + m_R v_k,$$

set $u = u_{k+1}^{int}$,

otherwise, if $i_u^k > 3$ set $u = u_k/2$.

Set

$$u_{k+1} = \max\left\{u, u_k/10, u_{min}\right\},$$
$$i_u^{k+1} = \max\left\{i_u^k + 1, 1\right\}.$$

If $u_{k+1} \neq u_k$ set $i_u^{k+1} = 1$. STOP.

5.4 Algorithm

Now we can present the Proximal Bundle method for nonconvex unconstrained optimization.

Algorithm 5.3: **Algorithm Proximal Bundle.**

Step 0*(Initialization):* Select some starting point $x_1 \in \mathbb{R}^n$, an accuracy tolerance $\varepsilon_s > 0$, the maximum number of stored subgradients $M_g \geq 2$, an initial weight $u_1 > 0$, a weights lower bound u_{min}, line search parameters $m_L \in (0, \frac{1}{2})$, $m_R \in (m_L, 1)$ and $\bar{t} \in (0, 1]$. Set the distance measure parameter $\gamma_f > 0$ (for convex function $\gamma_f = 0$), iteration counter $k = 1$, count the line search parameter $\zeta = 1 - \frac{1}{2} \cdot \frac{1}{1-m_L}$ and initialize variables:

$$\begin{cases} y_1 = x_1, & p_f^0 = g_1^f \in \partial f(y_1), \quad f_p^1 = f_1^1 = f(y_1) \\ s_f^1 = s_1^1 = 0, & J_1 = \{1\}. \end{cases}$$

Step 1*(Direction finding):* Find multipliers λ_p^k,λ_j^k for $j \in J_f^k$ by solving this dual problem

$$\begin{cases} minimize & \frac{1}{2u_k} \left\| \sum_{j=1}^{k} \lambda_j g_j^f + \lambda_p p_f^{k-1} \right\|^2 + \sum_{j=1}^{k} \lambda_j \alpha_{f,j}^k + \lambda_p \alpha_{f,p}^k \\ subject\ to & \sum_{j=1}^{k} \lambda_j + \lambda_p = 1 \\ & \lambda_j, \lambda_p \geq 0. \end{cases}$$

where

$$\begin{aligned} \beta_{f,j}^k &= \max\left\{ |f(x_k) - f_j^k|, \gamma_f (s_j^k)^2 \right\} \ \text{ for all } \ j \in J_f^k, \\ \beta_{f,p}^k &= \max\left\{ |f(x_k) - f_p^k|, \gamma_f (s_f^k)^2 \right\}. \end{aligned}$$

Also set

$$\begin{aligned} p_f^k &= \sum_{j \in J_f^k} \lambda_j^k g_j^f + \lambda_p^k p_f^{k-1}, \\ \tilde{f}_p^k &= \sum_{j \in J_f^k} \lambda_j^k f_j^k + \lambda_p^k f_p^k, \\ \tilde{s_f^k} &= \sum_{j \in J_f^k} \lambda_j^k f_j^k + \lambda_p^k s_f^k, \\ \tilde{\beta}_{f,p}^k &= \max\left\{ |f(x_k) - \tilde{f}_p^k|, \gamma_f (\tilde{s}_f^k)^2 \right\} \end{aligned}$$

and $d_k = -\frac{1}{u_k} p_k$

Step 2*(Stopping criterion):* Set

$$w_k = \frac{1}{2}\|p_k\|^2 + \tilde{\beta}^k_{f,p}$$

If $w_k \leq \varepsilon_s$ then STOP.

Step 3*(Line search):* Using line search algorithm mentioned in the section 5.2 find step size $t^k_L \in [0,1]$ and $t^k_R \in [t^k_L, 1]$. Set

$$x_{k+1} = x_k + t^k_L d_k \qquad \text{and} \qquad y_{k+1} = x_k + t^k_R d_k.$$

Step 4*(Linearization update):* Calculate the values

$$\begin{aligned} f^{k+1}_j &= f^k_j + t^k_L (g^f_j)^T d_k, \ \text{ for } \ j \in J^k_f, \\ s^{k+1}_j &= s^k_j + t^k_L \|d_k\|, \ \text{ for } \ j \in J^k_f, \\ f^{k+1}_p &= \tilde{f}^k_p + t^k_L (p^f_f)^T d_k, \\ s^{k+1}_f &= \tilde{s}^k_f + t^k_L \|d_k\|. \end{aligned}$$

Evaluate $g^f_{k+1} \in \partial f(y_{k+1})$ and set

$$\begin{aligned} f^{k+1}_{k+1} &= f(y_{k+1}) + (t^k_L - t^k_R)(g^f_j)^T d_k, \\ s^{k+1}_{k+1} &= (t^k_R - t^k_L)\|d_k\|. \end{aligned}$$

Step 5*(Weight updating):* Select u_{k+1} by the algorithm in the section 5.3.

Step 6*(Updating):* Set $J^{k+1}_f = J^k_f \cup \{k+1\}$, but if $J^{k+1}_f > M_g$, then $J^{k+1}_f = J^{k+1}_f \setminus \{\min j |\ j \in J^{k+1}_f\}$. Increase k by 1 a go on ***Step 1***.

5.4.1 Convergence analysis

- **Convex function** f

  Theorem 5.4.1. Put

  $$X^* = \{x^* \in \mathbb{R}^n | f(x^*) \leq f(x) \text{ for all } x \in \mathbb{R}^n\}.$$

  If $X^* \neq \emptyset$ the $\{x_k\}$ converges to some $\overline{x} \in X^*$ as $k \to \infty$.
  If $X^* = \emptyset$ the $\{f(x_k)\}$ converges to some $\inf\{f(x)|x \in \mathbb{R}^n\} \in [-\infty; \infty)$.

- **Nonconvex function** f

  Theorem 5.4.2. Let f be weakly semi-smooth [1] and locally Lipschitz on $\mathbb{R}^n$. If f is bounded below and $\{x_k\}$ is bounded, then there exists a cluster point $\tilde{x}$ of the sequence $\{x_k\}$ such that $0 \in \partial f(\tilde{x})$.

Proof. Both of the proofs can be found in [6].

5.4.2 Algorithm implementation

We should show how we implement the *Step 1* in the Proximal Bundle method algorithm. This algorithm was implemented in MATLAB, which has many internal functions. In the direction finding part, we want to use `quadprog` so we have to modify the dual problem *(ADP)* into form

$$\min_x \frac{1}{2} x^T H x + f^T x,$$

that this `quadprog` function requires.

Compilation of matrix H and vectors x, f.

Lets rewrite $\frac{1}{2}\left\|\sum_{j=1}^{k} \lambda_j g_j + \lambda_p p_f\right\|^2$ into matrix notation.

$$\begin{aligned}\frac{1}{2}\left\|\sum_{j=1}^{k} \lambda_j g_j + \lambda_p p_f\right\|^2 &= \frac{1}{2}\left\|\sum_{j=1}^{k+1} \tilde{\lambda}_j \tilde{g}_j\right\|^2 = \\ &= \frac{1}{2}\left(\sum_{j=1}^{k+1} \tilde{\lambda}_j \tilde{g}_j\right)^T \left(\sum_{j=1}^{k+1} \tilde{\lambda}_j \tilde{g}_j\right) = \\ &= \frac{1}{2}\tilde{\lambda}^T G^T G \tilde{\lambda} = \frac{1}{2}\tilde{\lambda}^T (G^T G)\tilde{\lambda} = \\ &= \frac{1}{2}\tilde{\lambda}^T H \tilde{\lambda}\end{aligned} \tag{5.45}$$

[1] It means that the directional derivative $f'(x; v)$ exists for all $x \in \mathbb{R}^n$ and $v \in \mathbb{R}^n$.

So

$$G = \begin{bmatrix} p_f, & g_1, & g_2, & \ldots, & g_k \end{bmatrix}, \tag{5.46}$$

$$H = \frac{1}{u_k} G^T G, \tag{5.47}$$

$$\tilde{\lambda} = \begin{bmatrix} \lambda_p, & \lambda_1, & \lambda_2, & \ldots, & \lambda_k \end{bmatrix}^T \tag{5.48}$$

and

$$\tilde{\beta} = \begin{bmatrix} \beta_{f,p}, & \beta_{f,1}, & \beta_{f,2}, & \ldots, & \beta_{f,k} \end{bmatrix}^T. \tag{5.49}$$

Matrix G and vectors $\tilde{\lambda}$,$\tilde{\beta}$ have on its first position an aggregate element. Due to the limit of stored subgradients M_g, we have to solve how to rewrite the next elements of G,$\tilde{\lambda}$ and $\tilde{\beta}$. In every iteration, we test if size of matrix and vectors is higher then M_g. If it does, we delete the second column (or element) and add a new one at the end. If it doesn't, we don't delete anything.

Now we can rewrite the *(ADP)* problem into quadratic problem

$$\left.\begin{array}{ll} minimize & \frac{1}{2}\tilde{\lambda}^T H \tilde{\lambda} - \lambda^T \tilde{\beta} \\ subject\ to & A\tilde{\lambda} = B \\ & LB \leq \tilde{\lambda} \leq UB \end{array}\right\} \qquad \textit{(QP)}$$

where $A = [1, 1, \ldots, 1]$, $B = [1]$, $LB = [0, 0, \ldots, 0]^T$ and $UB = [\infty, \infty, \ldots, \infty]^T$.

6

Numerical experiments

After a detailed algorithm description in the previous chapter, we shall report some numerical experiments to show reliability of our code. The Proximal Bundle method was implemented in MATLAB 2012a. A summary of all the tests can be found in this part.

6.1 Tested functions

We have tested some traditional examples. Here we can see a summary of used functions.

F1 (Rosenbrock)

Although this function is smooth, it is often used as a test function.

Dimension	2
Cost function	$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$
General gradient at $\overline{x}$	$\partial f(\overline{x}) = \nabla f(\overline{x}) = \begin{bmatrix} 400x_1^3 - 400x_1x_2 \\ 200(x_2 - x_1^2) \end{bmatrix}$
Optimum point	$\overline{x} = [1; 1]$
Optimum value	$f(\overline{x}) = 0$
Starting point	$x_0 = [1.2; -0.3]$

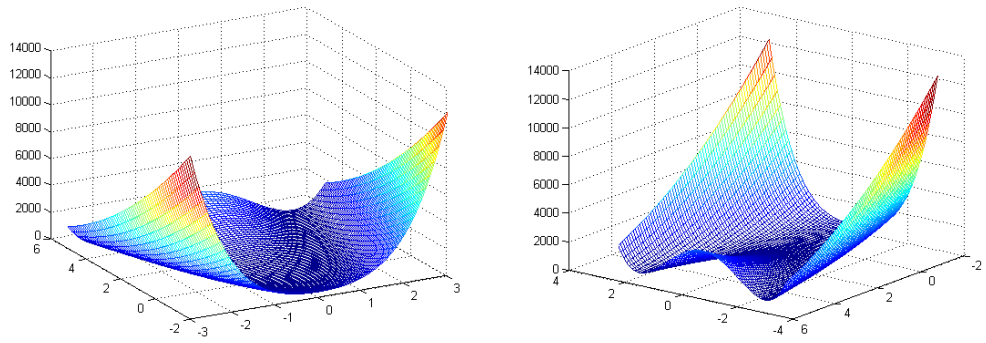

Figure 6.1: Graph of function $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$.

F2

Dimension	2
Cost function	$f(x) = \|x_1\| + \|x_2\|$
General gradient at $\overline{x}$	$\partial f(\overline{x}) = conv\,\{[1;1], [-1;1], [1;-1], [-1;-1]\}$
Optimum point	$\overline{x} = [0;0]$
Optimum value	$f(\overline{x}) = 0$
Starting point	$x_0 = [3;4]$

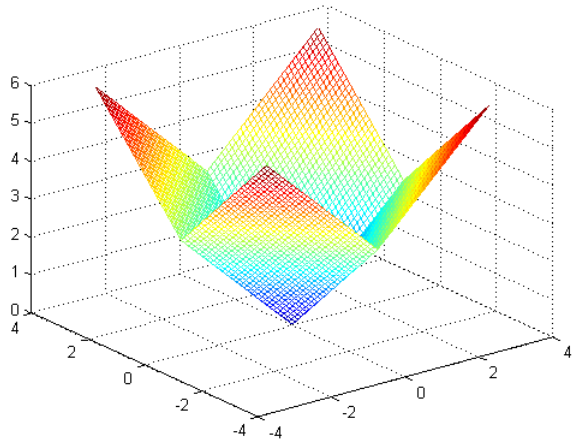

Figure 6.2: Graph of function $f(x) = |x_1| + |x_2|$.

F3 (Crescent)

Dimension	2
Cost function	$f(x) = \max\{x_1^2 + (x_2-1)^2 + x_2 - 1, \\ -x_1^2 - (x_2-1)^2 + x_2 + 1\}$
General gradient at $\overline{x}$	$\partial f(\overline{x}) = conv\left\{\begin{bmatrix} 2x_1 \\ 2x_2 - 1 \end{bmatrix}, \begin{bmatrix} -2x_1 \\ -2x_2 + 3 \end{bmatrix}\right\}$
Optimum point	$\overline{x} = [0; 0]$
Optimum value	$f(\overline{x}) = 0$
Starting point	$x_0 = [4; 4]$

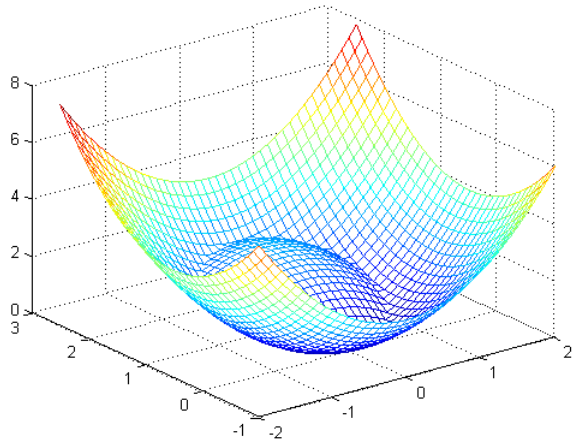

Figure 6.3: Graph of function $f(x) = \max\{x_1^2 + (x_2-1)^2 + x_2 - 1, -x_1^2 - (x_2-1)^2 + x_2 + 1\}$.

F4

Dimension	20
Cost function	$f(x) = \max\limits_{1 \le i \le 20}\{\lvert x_i \rvert\}$
General gradient at $\overline{x}$	$\partial f(\overline{x}) = conv\left\{[0, \cdots, \underset{i}{1}, \cdots, 0] \text{ where } 1 \le i \le 20\right\}$
Optimum point	$\overline{x} = [0, \cdots, 0]$
Optimum value	$f(\overline{x}) = 0$
Starting point	$x_0^i = i, \text{ for } 1 \le i \le 10$
	$x_0^i = -i, \text{ for } 11 \le i \le 20$

F5

Dimension	50
Cost function	$f(x) = 50 \max\limits_{1 \leq i \leq 50} \{x_i\} - \sum\limits_{i=1}^{50} x_i$
General gradient at $\overline{x}$	$\partial f(\overline{x}) = conv \left\{ [-1, \cdots, \underset{i}{49}, \cdots, -1] \text{ where } 1 \leq i \leq 50 \right\}$
Optimum point	$\overline{x} = [0, \cdots, 0]$
Optimum value	$f(\overline{x}) = 0$
Starting point	$x_i = i - 25.5$ for $i = 1..50$.

F6

Dimension	4
Cost function	$f(x) = \max\{f_1, f_1 + 10f_2, f_1 + 10f_3, f_1 + 10f_4\}$ $f_1 = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4$ $f_2 = x_1^2 + x_2^2 + x_3^2 + x_4^2 - x_1 - x_2 + x_3 - x_4 - 8$ $f_3 = x_1^2 + 2x_2^2 + x_3^2 + 2x_4^2 - x_1 - x_4 - 10$ $f_4 = x_1^2 + x_2^2 + x_3^2 + 2x_1 - x_2 - x_4 - 5$
General gradient at $\overline{x}$	$\partial f(\overline{x}) = conv \left\{ \begin{bmatrix} 2x_1 - 5 \\ 2x_2 - 5 \\ 4x_3 - 21 \\ 2x_4 + 7 \end{bmatrix}, \begin{bmatrix} 22x_1 + 5 \\ 22x_2 - 15 \\ 24x_3 - 11 \\ 22x_4 - 3 \end{bmatrix}, \begin{bmatrix} 22x_1 - 15 \\ 42x_2 - 5 \\ 24x_3 - 21 \\ 42x_4 + 3 \end{bmatrix}, \begin{bmatrix} 22x_1 + 15 \\ 22x_2 - 15 \\ 24x_3 - 21 \\ 2x_4 - 3 \end{bmatrix} \right\}$
Optimum point	$\overline{x} = [0; 1; 2; -1]$
Optimum value	$f(\overline{x}) = -44$
Starting point	$x_0 = [0; 0; 0; 0]$

6.2 Numerical results

We use the following abbreviations:

- **it** the number of iteration
- **n** the number of serious steps (short and long)
- $\mathbf{f}^*$ the final value of the cost function
- $\mathbf{x}^*$ the optimum point

The parameters had the following setings:

$m_L = 0.01$	$m_R = 0.5$	$\bar{t} = 0.01$
$u_{min} = 1\ 10^{-10}$	$\varepsilon = 10^{-3}, 10^{-6}$	$u_1 = \|g_1^f\|$

The convexity measure was:

$\gamma_f = 0$	$\mathbf{F2}, \mathbf{F4}, \mathbf{F5}, \mathbf{F6}$
$\gamma_f = 0.3$	$\mathbf{F1}$
$\gamma_f = 1$	$\mathbf{F3}$

The limit of stored subgradients had these values:

$M_g = 10$	$\mathbf{F1}, \mathbf{F2}, \mathbf{F3}, \mathbf{F6}$
$M_g = 200$	$\mathbf{F4}, \mathbf{F5}$

In the next tables 6.1 and 6.2, we summarize our results.

Precision	$\mathbf{10^{-3}}$			
Function	**it**	**n**	$\mathbf{f}^*$	$\mathbf{x}^*$
F1	21	18	$6.3668\ 10^{-5}$	$\begin{bmatrix} 0.9958 \\ 0.9909 \end{bmatrix}$
F2	5	4	$5.5511\ 10^{-17}$	$\begin{bmatrix} 0 \\ -5.5511\ 10^{-17} \end{bmatrix}$
F3	10	8	$2.3623\ 10^{-4}$	$\begin{bmatrix} -1.2613\ 10^{-15} \\ -2.3618\ 10^{-4} \end{bmatrix}$
F4	108	87	$2.9055\ 10^{-4}$	
F5	51	33	$1.3462\ 10^{-12}$	
F6	118	42	-43.9996	$\begin{bmatrix} -7.8958\ 10^{-4} \\ 1.0013 \\ 2.0002 \\ -0.9995 \end{bmatrix}$

Table 6.1: Results of Proximal Bundle methods - precision $\varepsilon = 10^{-3}$.

Precision	10^{-6}			
Function	**it**	**n**	$\mathbf{f}^*$	$\mathbf{x}^*$
F1	119	71	$1.6898\ 10^{-7}$	$\begin{bmatrix} 0.9996 \\ 0.9993 \end{bmatrix}$
F2	5	4	$5.5511\ 10^{-17}$	$\begin{bmatrix} 0 \\ -5.551110^{-17} \end{bmatrix}$
F3	26	13	$1.2681\ 10^{-8}$	$\begin{bmatrix} -7.5144\ 10^{-10} \\ -1.2681\ 10^{-8} \end{bmatrix}$
F4	161	130	$2.0839\ 10^{-7}$	
F5	51	33	$1.3462\ 10^{-12}$	
F6	238	76	-44	$\begin{bmatrix} 1.2025\ 10^{-15} \\ 0.9999 \\ 1.9999 \\ -1.0001 \end{bmatrix}$

Table 6.2: Results of Proximal Bundle methods - precision $\varepsilon = 10^{-6}$.

Convergence

In the next pictures, there is an illustration how a sequence $\{x_i\}_{i=1}^{\infty} \subset \mathbb{R}^n$ converges to the optimal point. The accuracy ε was set at 10^{-3}.

- **F1**

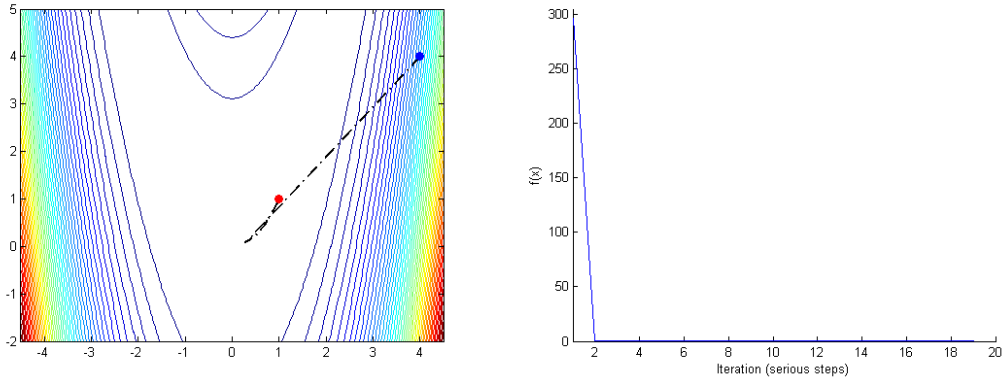

Figure 6.4: **F1** - Convergence of the sequence $\{x_i\}_{i=1}^{\infty} \subset \mathbb{R}^n$ and convergence to the optimal value $f(\overline{x})$.

- **F2**

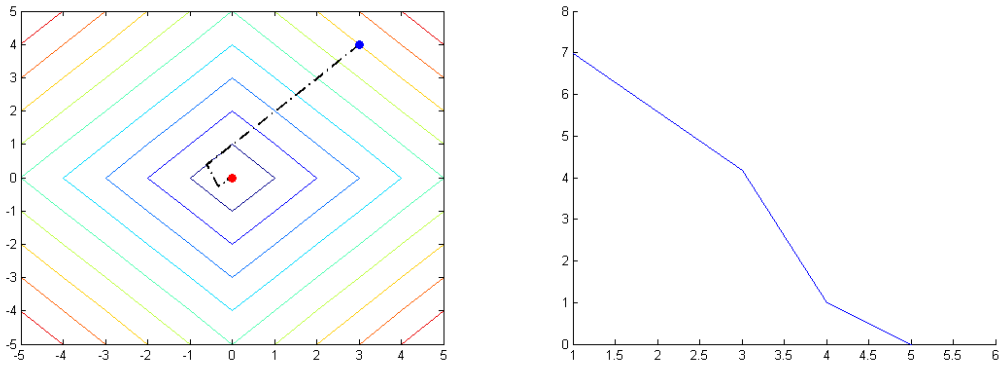

Figure 6.5: **F2** - Convergence of the sequence $\{x_i\}_{i=1}^{\infty} \subset \mathbb{R}^n$ and convergence to the optimal value $f(\overline{x})$.

- **F3**

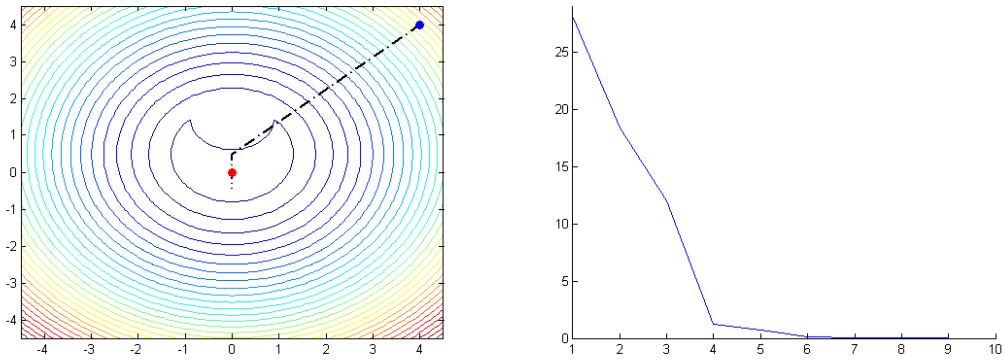

Figure 6.6: **F3** - Convergence of the sequence $\{x_i\}_{i=1}^{\infty} \subset \mathbb{R}^n$ and convergence to the optimal value $f(\overline{x})$.

Since the functions **F4**, **F5** and **F6** have dimensions greater then 2, we present here only the convergence to the optimal value $f(\overline{x})$.

- **F4**

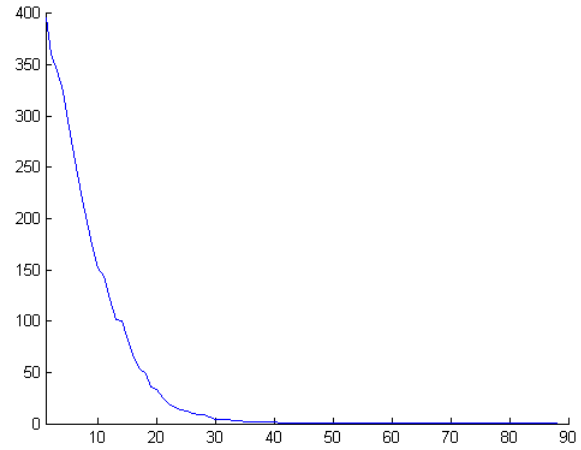

Figure 6.7: **F4** - Convergence to the optimal value $f(\overline{x})$.

- **F5**

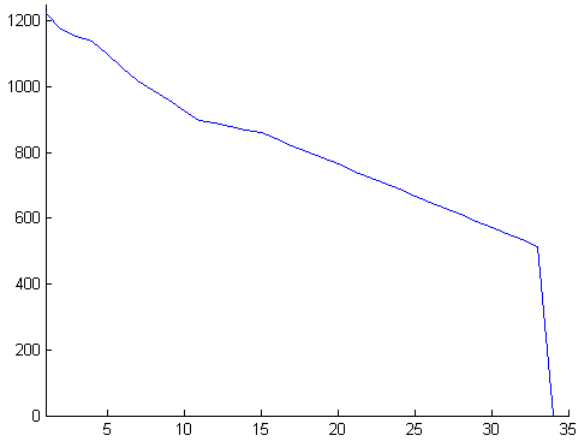

Figure 6.8: **F5** - Convergence to the optimal value $f(\overline{x})$.

- **F6**

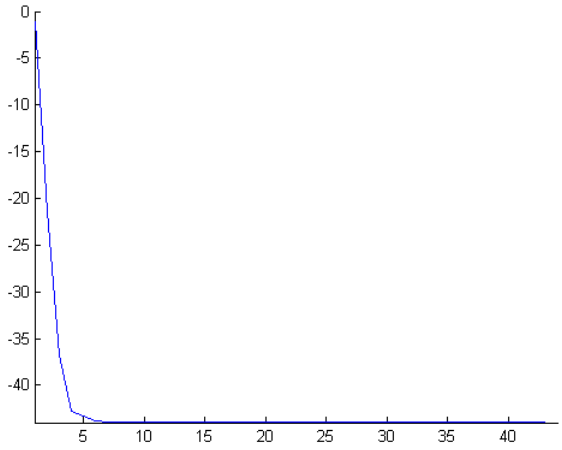

Figure 6.9: **F6** - Convergence to the optimal value $f(\overline{x})$.

7

Conclusion

This thesis dealt with the nonsmooth optimization methods. The aim was to introduce method called Proximal Bundle. First, we described optimization problem generally - we wrote about types of optimization programming and also about solution methods. After this short survey we followed up a nonsmooth analysis theory and we explained, why the classical solution methods are not useful when the cost function is not smooth. Therefore we introduced two methods, which are better to employ - Subgradient method and Proximal Bundle method.
The first of the mentioned method is very similar to Gradient descent or Newton's method, but instead of the gradient, we shall computed with the general one - the general gradient is defined as a convex hull of subgradients. We tested Subgradient method on example - we found a minimum of function $f(x) = |x_1| + |x_2|$ in two ways. First, the step size t_k was chosen a priori and then with the knowledge of the function value in optimal point $\overline{x} = [0; 0]$.
The second mentioned method is described more in detail. Two chapters are devoted to the Proximal Bundle method, especially to the algorithm analysis. Every step of the algorithm has its own section, where we wrote particularised characterization. To proof reliability of every step, we implemented the whole algorithm in MATLAB and we made several tests. We used some convex and nonconvex functions for testing. In the summary of all results, we could see the number of iterations, serious steps and the function value in optimal point.
Since we implemented algorithm for unconstrained optimization, we would like to continue with this work and expand it with some constraint, which should have a greater use: e.g. in a shape optimization.
We hope, that this thesis will help another people with a better understanding of this topic.

Bibliography

[1] DOSTÁL,Z., BEREMLIJSKI.P., *Metody optimalizace* [online]: skriptum. VŠB - Technical University of Ostrava, 2012. [quoted 2014-10-23]
Available on the internet: <http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/metody_optimalizace.pdf>.

[2] VONDRÁK,V., POSPÍŠIL,L., *Numerické metody I* [online]: skriptum. VŠB - Technical University of Ostrava, 2011. [quoted 2014-10-28]
Available on the internet: <http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/numericke_metody.pdf>.

[3] BOUCHALA,J., *Úvod do funkcionální analýzy* [online]: skriptum. VŠB - Technical University of Ostrava, 2012. [quoted 2014-10-28]
Available on the internet: <http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/uvod_do_funkcionalni_analyzy.pdf>.

[4] MÄKELÄ,M.M., NEITTAANMÄKI,P., *Nonsmooth Optimization, Analysis and Algorithms with Application to Optimal Control* [book]. World Scientific, Singapore, 1992. [quoted 2014-10-18].

[5] ZOWE,J., *Nondifferentiable Optimization* [book]. University of Bayreuth, Germany, 1985. [quoted 2014-10-18].

[6] OUTRATA,J., KOČVARA,M., ZOWE,J., *Nonsmooth Approach to Optimization Problems with Equilibrium Constraints, Nonconvex optimization and its applications* [book]. Kluwer Academic Publishers, 1998 [quoted 2015-1-4].

[7] KIWIEL,K.C., *Proximity Control in Bundle Methods for Convex Nondifferentiable Minimization* [article]. Polish Academy of Science, Warsaw, 1990. [quoted 2015-04-18].

Appendix A

CD description

CD contains these MATLAB m-files:

Subgradient method

SubgradMethod.m - an algorithm of implemented Subgradient method used for finding an optimal point

F1.m - tested function $f(x) = |x_1| + |x_2|$

subgrad_F1.m - subgradient of the tested function

Syntax: $[x, it] =$ SubgradMethod$(@F1, @subgrad_F1, min, x0, eps)$

Input:

$F1$	included on CD
$subgrad\_F1$	included on CD
min	an optimal point value (important for the second type of step size)
$x0$	an initial point
eps	required accuracy

Output:

x	an optimal point
it	number of iterations

Proximal Bundle method

linesearch.m - an algorithm of line search presented in the section 5.2

weight_update.m - an algorithm of weight update presented in the section 5.3

prox_bundle.m - an algorithm of Proximal Bundle method presented in the section 5.4

F1.m - F6.m - tested functions

subgrad_F1.m - subgrad_F6.m - subgradient of the tested functions

x0_F4.m, x0_F5.m - a routine returning an initial point for functions **F4** and **F5**

Syntax: $[x, it, f\_val, n] =$ prox_bundle$(@fun, @subgrad, x0, eps)$

Input:

fun	included on CD (*F1.m - F6.m*)
$subgrad\_F1$	included on CD (*subgrad_F1.m - subgrad_F6.m*)
$x0$	an initial point (for **F4** and **F5** included on CD)

Output:

x	an optimal point
it	number of iterations
f_val	a vector with function value at every x obtained in serious step at the first position - function value at the initial point at the last position - function value at the optimal point
n	number of serious steps

Appendix B

Algorithms

Subgradient method

```
function [X, it ] = SubgradMethod( F,subgrad_F, min, x0, my_eps
   )

x = x0;
it = 1;  %number of iteration
lambda = 1;  %step size coefficient 0 < lambda < 2;
alpha = 1/it;

gn = feval(subgrad_F,x);
dn = -gn/norm(gn);
xn = x + alpha * dn;

%while norm(feval(F,xn) - feval(F,x)) > my_eps
while norm(xn- x) > my_eps
    x = xn;
    gn = feval(subgrad_F,x);
    dn = -gn/norm(gn);
    %alpha = 1/it;     %a priori step size (1st type)
    alpha = lambda*(feval(F,x)-feval(F,min))/norm(gn); %(2nd
       type)
    xn = x + alpha * dn;
    it = it + 1;

end
    X = xn;
end
```

Listing B.1: Subgradient method

Proximal Bundle method

Line search

```
function [ tL, tR ] = linesearch( fun, subgr,mL,mR,tB,gamma_f,x,
   d,v )

%SETTINGS
tL = 0;
t = 1;
tU = 1;
zeta = 1 - 1/2*(1)/(1-mL);

while 1
    if  feval(fun, x + t*d) <= feval(fun,x) + mL*t*v
        tL = t;
    else
        tU = t;
    end

    if tL >= tB
        tR = tL;
        return
    else
        g = feval(subgr,x + t*d);
        beta = max(abs(feval(fun,x + tL*d) - feval(fun,x + t*d)
           +(t - tL)*g'*d ),gamma_f*(t - tL)^2*norm(d)^2 );
        if -beta + g'*d >=mR*v
            tR = t;
            return
        end
    end

    if tL == 0
        t = max(zeta*tU, (1/2 * tU^2 *v)/(tU*v + feval(fun,x) -
           feval(fun,x+t*d)));
    else
        t = 1/2 * (tL + tU);
    end

end
end
```

Listing B.2: Line search

Weight update

```
function [u_new ] = weight_update(f_yK1, f_X,m_R,uK, uInt, uMin,
   tL,v,betafK1,pB )

global I
global my_eps
u_old = uK;

if  0 < tL && tL < 1
    u_new = uK;
    return
end

    if tL <=0   %Step iii and after Step v  (Null step)
       my_eps = min(my_eps, pB);
       if (I < -3) && (betafK1 > max(my_eps, -10*v))
           u_old = uInt;
       end
       u_new = min(u_old, 10*uK);
       I = min(I-1,-1);
       if u_new ~= uK
           I = -1;
       end
    else  %Step iv  (Long serious step)
       my_eps = max(my_eps, -2*v);
       if (I >0) && (f_yK1 <= f_X + m_R*v)
           u_old = uInt;
       elseif I >3
           u_old = uK/2;
       end

       u_new = max([u_old, uK/10 ,uMin]);
       I = max(I+1,1);
       if u_new ~= uK
            I = 1;
       end
    end
end
```

Listing B.3: Weight update

Proximal Bundle

```
function [ x, it, f_val, m ] = prox_bundle(fun, subgrad,x0)

%PARAMETERS
tol = 1e-3;  % final tolerancy
M_g = 10; % maximum of stored subgradients
u = norm(feval(subgrad,x0)); % initial weight u > 0
u_Min = 1e-10; % minimal value for weight
m_L = 0.01; % line search parameter in (0; 0.5)
m_R = 0.5; % line search parameter in (m_L; 1)
t_b = 0.01; % line search parameter in (0; 1]
gamma_f = 0.3;% distance measure parameter (lambda_f = 0 if
   function is convex)

global I
global my_eps
I = 0;
my_eps = inf;

m = 1; %counter for serious steps
f_val(m) = feval(fun,x0); %function values after serious steps

dim = length(x0);

% QUADPROG parameters
G = zeros(dim,M_g+1);
beta = zeros(M_g+1,1);

A = ones(1,M_g+1);
B = 1;
LB = zeros(M_g+1,1);
UB = inf*ones(M_g+1,1);

%SETTINGS
x = x0; % starting point
y = x;
J(1) = 1;

f_p = feval(fun, y);
f_j = zeros(M_g+1,1);
f_j(1) = f_p;
s_f = 0;
s_j = zeros(M_g+1,1);
```

```
G(:,1) = feval(subgrad, y);  % equal to p_f
G(:,2) = feval(subgrad, y);

k = 1;

while 1
    l = length(J);

    for i = 2:l+1
        beta(i) = max(abs( feval(fun,x) - f_j(i-1) ),gamma_f *
           s_j(i-1)^2);
    end
    beta(1) = max(abs( feval(fun,x) - f_p ),gamma_f * s_f^2);

    H = (1/u)*(G(:,1:l+1))'*G(:,1:l+1);

    opt = optimset('Algorithm','active-set','Display','off');
    lambda = quadprog(H,beta(1:l+1),[],[],A(1:l+1),B,LB(1:l+1),
       UB(1:l+1),[],opt);


    G(:,1) = (G(:,1:l+1)*lambda);
    f(1,1) = f_p;
    s(1,1) = s_f;

    if J(1) == 1
        f(2:l+1,1) = f_j(1:l);
        s(2:l+1,1) = s_j(1:l);
    else
        f(2:l+1,1) = f_j(2:l+1);
        s(2:l+1,1) = s_j(2:l+1);
    end

    f_wp = (lambda)' * f;
    s_wf = (lambda)' * s;

    beta_wf = max( abs(feval(fun,x) - f_wp), gamma_f * s_wf^2 );

    d_k = (-1/u) * G(:,1);

    %STOPPING CRITERION
    w = ((norm(G(:,1))^2))/2 + beta_wf;
    if w <= tol;
```

```
    break
end

v = (-1/u)*(norm(G(:,1:l+1)*lambda) )^2 - ( lambda' * beta
   (1:l+1,:) );

%LINE SEARCH
[ t_L, t_R ] = linesearch( fun, subgrad,m_L,m_R,t_b,gamma_f,
    x, d_k, v );
x_prew = x;
x = x + t_L*d_k;
y = x_prew + t_R*d_k;

if t_L >=t_b;
    f_val(m+1) = feval(fun,x);
    m = m+1;
elseif  t_L > 0 && t_L < t_b;
    f_val(m+1) = feval(fun,x);
    m = m+1;
end

%LINEARIZATION UPDATE
for j = 1:l;
    if J(1) ==1
        f_j(j) = f_j(j) + t_L*(G(:,j+1))'*d_k;
        s_j(j) = s_j(j) + t_L*(norm(d_k))^2;
    else
        f_j(j) = f_j(j+1) + t_L*(G(:,j+1))'*d_k;
        s_j(j) = s_j(j+1) + t_L*(norm(d_k))^2;
    end
end
f_p = f_wp + t_L*(G(:,1))'*d_k;
s_f = s_wf + t_L*norm(d_k);

s_j(l+1) = (t_R- t_L)*norm(d_k);

%EDITING G (matrix of subgradients)
sizeG = size(G(:,1:l+1));
n = sizeG(2);
if n >= M_g +1
    G(:,2) = [];
    G(:,l+1) = feval(subgrad,y);
    f_j(l+1) = feval(fun,y) + (t_L - t_R)*(G(:,l+1))'*d_k;
else
```

```
    G(:,l+2) = feval(subgrad,y);
    f_j(l+1) = feval(fun,y) + (t_L - t_R)*(G(:,l+2))'*d_k;
    end

    %WEIGHT UPDATE
   pb = norm(G(:,1)) + beta_wf ;
   beta_FK = max((feval(fun,x) - f_j(l+1)), gamma_f*s_j(l+1));
   u_Int = 2*u*(1-(feval(fun,y) - feval(fun,x_prew))/v);
   u  = weight_update(feval(fun,y), feval(fun,x_prew),m_R,u,
      u_Int, u_Min,t_L,v,beta_FK,pb);

    %INDEX SET UPDATE
    if l+1 > M_g
        J(1) = [];
        J(l) = k+1;
    else
    J(l+1) = k+1;
    end

    k = k + 1;
end
 it = k-1;
 m = m - 1;
end
```

Listing B.4: Proximal Bundle